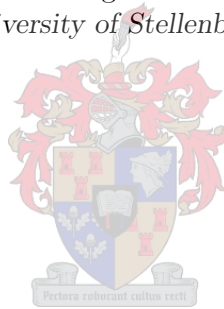


A Relation-based Approach to Engineering Management Systems

by

Jacobus Alexander van Breda Strasheim

*Dissertation presented for the degree of Doctor of Philosophy at the
University of Stellenbosch*



Department of Civil Engineering
University of Stellenbosch
Private Bag X1, Matieland 7602, South Africa

Promoter: Dr G. C. van Rooyen
Department of Civil Engineering
University of Stellenbosch

March 2007

Declaration

I, the undersigned, hereby declare that the work contained in this dissertation is my own original work and that I have not previously, in its entirety or in part, submitted it at any university for a degree.

Signature:

J. A. v. B. Strasheim

Date:

Abstract

A Relation-based Approach to Engineering Management Systems

J. A. v. B. Strasheim

Department of Civil Engineering

University of Stellenbosch

Private Bag X1, Matieland 7602, South Africa

Dissertation: PhD (Eng)

March 2007

The primary goal of this thesis is to indicate how systems theory and engineering process modelling can be applied to provide models for consulting engineering service business enterprises. The typical management systems used for these businesses are investigated to determine the application of systems and process models.

The motivation for this study is based on the fact that integrated management systems for consulting engineering practices are presently based on selective business analysis and process modelling that has evolved over time, as reported in a survey and study by Smit [110]. Furthermore, current engineering management systems are simply computer implementations of management procedures based on techniques that were developed to solve problems in the absence of the computational capabilities of the modern computer. To rectify this, a fundamental approach to analyse the business and management functions using systems theory and engineering process modelling techniques is required, which has not been attempted to date. This study develops and demonstrates the application of fundamental analysis in consulting engineering enterprise management and reviews advantages that can be obtained from using this approach.

It is shown that the mathematical Algebra of Relations and associated Graph Theory provide the mathematical basis on which management problems can be treated systematically. Since these fields of mathematics are well developed and very broad, the essential parts of the theories are identified. Thereupon, the application of the very abstract mathematical concepts to two important and typical engineering management problems are developed, which represents the core contribution of the dissertation.

The study is developed and presented in two parts and an addendum:

1. The first part provides an overview of the necessary mathematical theory required to support development of business models.
2. Management systems theory and relation- and graph theory-based engineering process modelling techniques are applied in this part to build generic enterprise models and data processing models. These models provide inputs for the management processes of professional service business enterprises. The outcome of the modelling and analysis is a set of database models with reporting functionality, to be used in the management process. A demonstration of technology available for development of the models and techniques, described in the previous part, is undertaken in this part. Generic implementations of database models and reporting techniques for systems which deal with management data in a consulting engineering business are developed, described and demonstrated.
3. In the addendum to the study, typical models and system functionality needed to support the management functions of the consulting engineering service business are identified. These management functions include:
 - Business strategy and long term planning
 - Marketing and promotion

- Finance, including bookkeeping and auditing
- Personnel
- Facilities management and document management
- Logistics, i.e., management of resources required for the business to operate
- Knowledge management
- Production management, i.e., management of the execution of project work
- Administration
- Risk management

Production management can use the engineering process model approach, modelling the management of tasks, persons, datasets and tools as these are applied to the consulting engineering business.

Sample subsystems to support selected management functions are identified and analysed. The integration of these systems with commercially available systems to support accounting and management reporting can follow from this analysis.

The study contributes to the body of knowledge in the field of engineering management by providing insights into the application of a specific branch of mathematics to provide fundamental solutions to engineering management problems. It also shows how these solutions are mapped to the computer, and describes available information techniques and technology to support the mapping. The outcome is a document setting out the theory required to develop robust enterprise management systems, the development and demonstration of technology required to do this and, as an addendum, a high level specification of business and management system functionality required for the professional engineering service business.

Uittreksel

'n Relasie-baseerde Benadering tot Ingenieurs-bestuurstelsels

J. A. v. B. Strasheim

Departement Siviele Ingenieurswese

Universiteit van Stellenbosch

Privaatsak X1, Matieland 7602, Suid-Afrika

Proefskrif: PhD (Ing)

Maart 2007

Die hoofdoel van hierdie tesis is om aan te toon hoe stelselteorie en ingenieursprosesmodellering toegepas kan word om modelle van raadgewende ingenieurs diens-besighede te lewer. Die tipiese bestuurstelsels van hierdie besighede word ondersoek om die toepassing van die stelsels en prosesmodelle te bepaal.

Die motivering vir die studie is baseer op die feit dat geïntegreerde bestuurstelsels vir raadgewende ingenieurspraktyke gebaseer is op selektiewe besighedsanalise en prosesmodelle, wat oor tyd ontwikkel het soos gerapporteer in 'n studie en opname deur Smit [110]. Die ingenieursbestuurstelsels wat tans in gebruik is, is slegs rekenaar implementerings van bestuursprosedures wat op tegnieke baseer is wat ontwikkel is in die afwesigheid van die berekeningsvermoë van die moderne rekenaar. Om hierdie probleem aan te spreek word 'n fundamentele benadering tot analise van die besigheds – en bestuursfunksies waarin stelselteorie en ingenieursproses modellering tegnieke gebruik word benodig. So 'n benadering is tot datum nie aangepak nie. Hierdie studie ontwikkel en demonstreer die toepassing van fundamentele analise vir ondernemingsbestuur van raadgewende ingenieursbesighede en verskaf 'n oorsig van die voordele wat uit hierdie benadering behaal kan word.

Dit word aangetoon dat die wiskundige Algebra van Relasies en geassosieerde Grafiekteorie die wiskundige basis verskaf waarop Bestuursprobleme sistematies aangepak kan word. Aangesien hierdie afdelings van die wiskunde goed ontwikkel is en 'n wye veld dek word die essensiële dele van die teorie identifiseer. Daarna word die abstrakte konsepte toegepas op twee belangrike tipiese ingenieursbestuur probleme as die kern bydrae van die verhandeling.

Die studie word in twee dele en 'n addendum ontwikkel en aangebied:

1. Die eerste deel verskaf 'n oorsig van die nodige wiskundige teorie wat benodig word om die ontwikkeling van die besighedsmodelle te ondersteun.
2. Basiese bestuurstelselteorie en relasie–baseerde ingenieursproses modellering tegnieke word in hierdie deel toegepas om generiese besighedsmodelle en data-verwerkingsmodelle te ontwikkel vir professionele diensondernemings. Hierdie modelle kan insette lewer aan die bestuursprosesses van die onderneming. Die uitkoms van die modelle en analise is 'n stel van databasis modelle met verslag funksionaliteit wat in die bestuursproses gebruik kan word. 'n Demonstrasie van die tegnologie wat beskikbaar is vir die ontwikkeling van modelle en tegnieke wat in die vorige afdeling beskryf is, word in hierdie deel aangebied. Generiese implementering van databasismodelle en verslagtegnieke vir stelsels wat bestuursdata in 'n raadgewende ingenieursbesigheid verwerk, word ontwikkel, beskryf en gedemonstreer.
3. In die addendum tot die studie word tipiese modelle en stelselfunksionaliteit identifiseer wat die bestuursfunksies van die raadgewende ingenieursbesigheid kan ondersteun. Hierdie bestuursfunksies sluit in:

- Besigheidstrategie, beleid en langtermyn beplanning

- Bemarking en promosie
- Finansies, met rekeningkunde en oudit
- Personeel
- Fasiliteitsbestuur en dokumentbestuur
- Logistiek - dit is die bestuur van hulpbronne benodig vir die bedryf van die besigheid
- Bestuur van kennis en kundigheid
- Produksie bestuur - dit is die bestuur en uitvoer van projekwerk
- Administrasie
- Risiko bestuur

Produksie bestuur kan die ingenieursprosesmodel benadering direk gebruik om die generiese konsep van take, persone, data stelle en gereedskap, soos toegepas op 'n raadgevende ingenieursonderneming, te benut.

Tipiese substelsels van geselekteerde bestuursfunksies word geïdentifiseer en ge-analiseer. Die integrasie van hierdie substelsels met kommersieel beskikbare stelsel kan benut word om rekeningkundige en bestuursverslagdoening te ondersteun.

Die studie dra by tot die beskikbare kennis in die veld van ingenieursbestuur deur die toepassing van 'n spesifieke afdeling van wiskunde om insigte te verskaf in die fundamentele oplossings vir ingenieursbestuur probleme. Dit toon ook aan hoe hierdie oplossings op die rekenaar afgebeeld word en beskryf beskikbare informasie verwerkings tegnieke en tegnologie om die afbeelding te ondersteun. Die uitkoms is 'n dokument wat die teorie benodig om deurgronde besigheidsbestuurstelsels te ontwikkel uiteensit, die tegnologie om dit te implementeer ontwikkel en demonstreer en, as 'n addendum, 'n hoë vlak spesifikasie van die besigheidstelsel funksionaliteit wat deur die professionele ingenieursdiensbesigheid benodig word, bevat.

Acknowledgements

The contribution of my study leader Dr. Gert van Rooyen, is appreciated. He has been instrumental in developing the field of study of Engineering Information Systems (German *Bauinformatik*) at the University of Stellenbosch.

This document was prepared using the MikTeX L^AT_EX document preparation system. References were made to works by inter alia Kopka and Daly [69], Lamport [70], Goossens et al. [46] as well as the various extended World Wide Web references on the software.

I would like to express my sincere gratitude to the following people and organisations:

- The MikTeX software was downloaded from Schenk [109].
- The work of Danie Els of the Department of Mechanical Engineering in developing and publishing L^AT_EX templates for University of Stellenbosch thesis documents - this is greatly appreciated.
- The reference database manager supplied by JabRef [65] . The underlying format of the data is in BiBTeXformat.
- Google, [44], for providing very valuable search facilities for internet world wide web resources.
- Wikipedia, [126], the open online encyclopaedia, for making available well organised and researched information on selected topics dealt with in this document.

Dedications

I would like to dedicate this document to my educators.

- Kindergarten and Grade 1 and 2 - Mrs. Schwemmer.
- Menlopark Primary School - Mr. Venter (principal), Mrs. van Rooyen, Mrs. Terreblanche, Mr. Breedt, Mr. de Villiers.
- Menlopark High School - Mr. van Zijl (principal), Mrs Vorster (Afrikaans), Mr Schroeder(English), Ms Du Toit(Mrs Puddo)(Science), Mr. Goris (Latin), Mr. Visser (Mathematics and Mechanics), Mr. van der Nest(History), Mr. Booysen (Geography), Mr. Cillers and Mr. Deneyschen(Physical Training).
- University of South Africa - Sciences - Prof. Rund.
- University of Pretoria - Civil Engineering - Prof. De Vos, Prof. Rooseboom, Prof. von Willigh, Dr. Hopkins, Mr. Wolmarans.
- University of Stellenbosch - Civil Engineering - Prof. du Preez, Prof. Louw, Mr. Vos, Mr. Sippel, Prof. Hugo, Prof. Pahl.
- University of Stellenbosch - Graduate School of Business - Prof. Gevers, Mr. Erasmus, Prof. Archer, Prof. Hamman.
- Geustyn, Forsyth & Joubert Inc., Consulting Engineers - Mr. Geustyn, Mr. du Toit, Mr. Krige and Mr. Kapp.

My father, Albertus Strasheim, mother Ina Strasheim as well as my wife, Helene and children Albert, Ingrid and Marike as well as my parents in law Gerrit and Lucille de Wet also indirectly contributed to my education.

Note: The names above were recalled from memory and do not constitute a complete list. Apologies to any person who feels that he or she needs to be on this list!

Contents

Declaration	i
Abstract	ii
Uittreksel	iv
Acknowledgements	vi
Dedications	vii
Contents	viii
List of Figures	xix
List of Tables	xxiii
Nomenclature	xxv
1 Introduction	1
I Management systems theory and engineering process modelling techniques	2
2 Overview of Part I	3
3 Set Theory	4
3.1 Introduction	4
3.2 Sets, Elements of Sets and Subsets	4
3.2.1 Definition of a set	4
3.2.2 Defining elements of sets	4
3.2.3 Defining families of elements	5
3.2.4 Universal and existential quantifiers	5
3.2.5 Equality of sets	5
3.2.6 Subsets of a set	5
3.2.7 Comparable sets	5
3.2.8 Sets of sets	5
3.2.9 Power set	5
3.2.10 Universal set	6
3.2.11 Disjoint sets	6
3.2.12 Partition of a set	6
3.3 Set Operations	6
3.3.1 Union of sets	6
3.3.2 Intersection of sets	6
3.3.3 Set difference	6
3.3.4 Complement of a set	6
3.4 Sets of Numbers	7
3.4.1 Integer numbers	7
3.4.2 Natural numbers	7
3.4.3 Rational and irrational numbers	7

3.4.4	Real numbers	7
3.4.5	Number inequalities	7
3.4.6	Absolute value	8
3.4.7	Intervals on sets of numbers	8
3.4.8	Bounded and unbounded sets of numbers	8
3.5	MATLAB implementation of Set Operations	9
4	Relations and Mappings	10
4.1	Introduction	10
4.2	Ordered pair	10
4.3	Cartesian product	10
4.4	Unary relations	10
4.5	Binary relations	11
4.6	Heterogeneous binary relation	11
4.7	Homogeneous binary relation	11
4.8	Properties of relations	11
4.9	Totality of a relation on A and B	11
4.10	Uniqueness of a relation on A and B	12
4.11	Relational diagram	12
4.12	Types of relations	13
4.12.1	Identity relation	13
4.12.2	Inverse relation	13
4.12.3	Composition	13
4.12.4	Equivalence relation	13
4.12.5	Equivalence class	13
4.12.6	Partitioning by equivalence	13
4.12.7	Quotient set	14
4.13	Mappings	14
4.13.1	Mapping notation	14
4.13.2	Image of an element	14
4.13.3	Arrow diagram	14
4.13.4	Types of mappings	15
4.14	Order relations and ordered sets	16
4.15	Countability and cardinality	16
4.15.1	Cardinal numbers and finite and infinite sets	16
4.15.2	Operations on cardinal numbers	17
4.15.3	Countable sets and properties of countable sets	17
4.15.4	Comparison of and ordering of cardinal numbers	17
4.15.5	Cardinality of the set of real numbers	18
4.15.6	Cardinality of Cartesian products of the set of real numbers	18
4.16	Closure of a homogeneous binary relation	18
4.16.1	Reflexive closure	18
4.16.2	Symmetric closure	18
4.16.3	Powers of a relation	18
4.16.4	Stability index	18
4.16.5	Transitive closure	19
4.16.6	Reflexive transitive closure	19
4.16.7	Reflexive symmetric transitive closure	19
4.17	Algebra of homogeneous binary relations	19
4.17.1	Graphical representation	19
4.17.2	Special relations	20
4.17.3	Equality and inclusion	20
4.17.4	Binary operations	20
4.18	MATLAB implementation of Relation Operations	21

5	Graph Theory	22
5.1	Introduction	22
5.2	Graphs and Directed graphs	22
5.3	Graphs	22
5.4	Graph isomorphism	22
5.5	Subgraphs	22
5.6	Directed graphs	23
5.6.1	Definition of a directed graph	23
5.6.2	Properties	23
5.7	Degrees, indegrees and outdegrees	23
5.7.1	Equality and inclusion	24
5.7.2	Adjacency matrix graph representation	24
5.8	Graph representation and manipulation	25
5.8.1	Adjacency matrices	25
5.8.2	Incidence matrices	25
5.9	Structure of graphs	27
5.9.1	Paths and cycles in directed graphs	27
5.9.2	Connectedness of directed graphs	30
5.9.3	Acyclic graphs	34
5.9.4	Simple acyclic graphs and trees	36
5.10	Rooted graphs and rooted trees	37
5.10.1	Root	37
5.10.2	Rooted graphs	37
5.10.3	Acyclic rooted graphs	37
5.10.4	Rooted trees	37
5.10.5	Forest of rooted trees	37
5.10.6	Search tree	37
5.11	Depth-first search	38
5.11.1	Depth-first search for trees and forests	38
5.11.2	Pre-order and post-order numbering	38
5.11.3	Classification of edges	38
5.11.4	Depth-first search algorithm	38
5.11.5	Depth-first search example	39
5.12	MATLAB implementation of basic graph functionality	41
6	Systems Theory	42
6.1	Introduction	42
6.2	Systems concepts and terminology	42
6.2.1	System structure	42
6.2.2	System function and behaviour	43
6.2.3	Control of systems	45
6.2.4	System performance measurement	46
6.2.5	Types of systems	46
6.2.6	Classification of systems	47
6.3	System laws	47
6.4	Formal specification of systems	47
6.4.1	Formal system structure definition	47
6.4.2	Formal system function definition	50
6.4.3	Basic example of formal system structure and function modelling	51
6.5	MATLAB implementation of basic system function example	52
6.6	System analysis	52
6.7	System design	53
6.8	Business enterprises and business enterprise systems	54
6.8.1	Business systems - structural models	54
6.8.2	The production process	56
6.8.3	Business systems supporting business operations	56
6.8.4	Business management systems - structural models	56
6.9	Formulation of required business model structure and functionality	58

6.9.1	Typical business object classification structures	58
6.9.2	Conceptual model of system functionality to process business objects	58
7	Relational Database Theory	61
7.1	Introduction	61
7.2	Database System Concepts and Architecture	61
7.2.1	Data Models, Schemas and Instances	61
7.2.2	DBMS Architecture	62
7.2.3	Data Modelling Techniques	63
7.3	Relational Data Model, Constraints and Relational Algebra	65
7.3.1	Relational Model Concepts	65
7.3.2	Tabular representation of a relation	66
7.3.3	Set theoretic Formulation of a Relational Database	66
7.3.4	The structural properties and characteristics of a relation	67
7.4	Candidate keys and Primary key of a Relation	67
7.5	Prime Attributes	68
7.6	Foreign Keys	68
7.6.1	Properties and characteristics of keys of relations	68
7.7	Key constraints	69
7.8	Functional Dependencies in Relations of Relational Databases	69
7.8.1	Full Functional Dependencies	70
7.8.2	Partial Functional Dependencies	70
7.8.3	Transitive Functional Dependencies	70
7.9	Database normalisation and Design	70
7.10	The data model	70
7.11	Insertion, Deletion and Update Operations on Relations	70
7.11.1	Inserting a Tuple into a Relation table	70
7.11.2	Deleting a Tuple from a Relation table	71
7.11.3	Updating a Tuple of a Relation Table	71
7.12	Specification of Attribute Domains	71
7.13	Relational Algebra, Calculus and Relational Operations	71
7.13.1	The Selection Operation	72
7.13.2	The Projection Operation	72
7.13.3	Tuple Concatenation Operation	72
7.14	Set Operations on Relations	73
7.14.1	Set Union Operation	73
7.14.2	Set Difference Operation	73
7.14.3	Set Intersection Operation	73
7.14.4	Set Cartesian Product Formation Operation	73
7.14.5	The Join Operation	74
7.14.6	Relation and Attribute Rename Operation	76
7.14.7	Grouping and Aggregation Operations	76
7.15	Relational Calculus	77
7.15.1	Tuple Relational Calculus	77
7.15.2	Domain Relational Calculus	78
7.15.3	Relational Algebra and Relational Calculus	79
7.16	Structured Query Language (SQL)	79
7.17	The database normalisation process	79
7.17.1	The First Normal Form	80
7.17.2	The Second Normal Form	81
7.17.3	The Third Normal Form	81
7.17.4	The Boyce-Codd Normal Form	82
7.17.5	The Fourth Normal Form	82
7.18	Database Transaction Processing	82
7.19	Additional reference material	82

II Management models and techniques - development technology demonstration	84
8 Overview of Part II	85
9 Relational Algebra MATLAB Tools	86
9.1 MATLAB boolean matrix relational algebra package (toolbox)	86
9.2 MATLAB Relational Algebra Tools Code	86
10 Literal String Processing MATLAB Functionality	89
11 Engineering Process Model	90
11.1 Introduction	90
11.2 Engineering process model, components and relations	90
11.2.1 Components of the model	90
11.2.2 Relations in the set of Tasks	92
11.2.3 Step schedule of tasks	92
11.2.4 Relations in the sets of Persons, Tools and Datasets	92
11.2.5 Order relation in the set of Tasks	92
11.3 Specification of the process model	94
11.3.1 Task-Dataset relationships	94
11.3.2 Task-Person relationships	95
11.3.3 Dataset-Tool relationships	95
11.4 Example A: Consulting engineering business process model	96
11.5 Relations computed from specified process model relations	97
11.5.1 Relations deduced by transposing the specified relations	97
11.5.2 Relations deduced by forming the union of all three Dataset-task relations	97
11.5.3 Relations computed between persons and datasets	100
11.5.4 Relations deduced by forming the union of the person - data and data - person relations	100
11.5.5 Relations computed using relations specified between persons and tools used by persons	100
11.5.6 Relations computed using relations specified between tasks and tools used to execute tasks	100
11.5.7 Relations deduced from dataset requires tool relation	101
11.5.8 Computing the logical sequence of tasks	101
11.5.9 Computing person loading	101
11.5.10 Computing tool loading	101
11.5.11 Computing dataset history	102
11.6 Process task specification reporting for the process model	102
11.7 Example B: Data evolution status value processing for process model	103
11.7.1 Process model set specification	103
11.7.2 Person- Task relation specification	104
11.7.3 Task - Data specification	104
11.7.4 Computed basic relations	104
11.7.5 Computing task sequence	104
11.7.6 MATLAB implementation of process model with status settings	105
11.7.7 MATLAB code for Engineering Process Model Example	105
11.8 Data file formats	105
11.9 Engineering Process Model - Figures	106
12 Representing and Processing management structure using graph applications	127
12.1 Introduction	127
12.2 Typical Management Reporting Tree Structure	127
12.3 Testing of the logic and integrity of the management structure	128
12.4 Converting from adjacency matrix format to adjacency list format	129
12.5 Depth first search and tree structure for a given graph	129
12.6 Inserting sub management structures into larger structures	130
12.7 Extracting subgraphs of vertices linked to a selected vertex	130

12.8 Management/ reporting structure - tree analysis examples	130
12.8.1 Basic example	130
12.8.2 Larger more realistic example	132
12.9 Determining connectivity of vertices in graphs to determine tree vertex links for roll up of reports	133
12.10 Report data roll up using adjacency matrices	133
13 Engineering Process Model: Database development, processing and report generation	139
13.1 Introduction	139
13.2 Database demonstration system overview	139
13.2.1 Client-server configuration	139
13.2.2 Network configuration	139
13.3 Defining the database structure	140
13.3.1 Azzurri Clay XML DTD Specification file	141
13.3.2 Database setup SQL statements	141
13.3.3 PostgreSQL Database Reference	141
13.3.4 Populating the database with data	142
13.3.5 Server database verification	142
13.4 Microsoft Access Database Reference	142
13.4.1 Access data import process	142
13.4.2 Database schema definition file	142
13.4.3 Database program to import data from database server	142
13.5 Reporting using Microsoft Access	144
13.6 Reporting using Microsoft Excel	144
13.7 PLEP application program for Engineering Process Model	146
13.8 Importing database data using the Java JDBC-ODBC bridge	146
13.9 SQL Programming for reports and SQL functionality used	146
13.9.1 PostgreSQL conversion functions	146
13.9.2 Microsoft Access data conversion	149
13.9.3 SQL Query Processing tips	149
13.9.4 SQL Queries for S-Curve presentation	152
13.10 Using Microsoft Data Access Pages	153
13.11 Conclusion and recommendations	153
III Conclusion	155
14 Conclusions and Recommendations	156
14.1 Conclusions	156
14.2 Recommendations	156
IV Addendum: Identification of system functionality to provide support for management functions	157
15 Overview of Part IV	158
16 Business strategy, long term planning and general management	161
16.1 Business Strategy Concepts and Strategy development	161
16.1.1 Elements of a business which reflect strategy	161
16.1.2 Physical indicators of the direction and 'look' of an enterprise	161
16.1.3 Strategic areas comprising an organisation	162
16.1.4 Maintaining a strong and healthy strategy	162
16.1.5 Articulating the business concept of the enterprise	163
16.1.6 Operational objectives	163
16.1.7 Developing strategic business models	163
16.2 General Management	164
16.2.1 Mechanical aspects of general management	164

16.2.2	Dynamic aspects of management - activities and processes	165
16.2.3	Communication	167
16.3	Conclusion and recommendation	167
17	Marketing, promotion and public relations management	168
17.1	Introduction to professional services marketing management	168
17.2	Differences between consumer product marketing and professional services marketing . . .	168
17.3	The nature of professional services marketing	169
17.3.1	Business models for marketing management	170
17.3.2	Business objects relating to marketing management	170
17.3.3	Business processes relating to marketing management	171
17.4	Marketing investigation, environmental scanning and forecasting	171
17.4.1	Corporate/enterprise requirements	172
17.4.2	Environmental scanning and forecasting	172
17.4.3	Marketing research and market research	172
17.4.4	Market segmentation	172
17.5	Strategic planning for marketing	172
17.6	Marketing activity planning and budgeting	173
17.7	Project phases	173
17.8	Organising for marketing	173
17.9	Marketing leading	174
17.10	Coordinating of marketing activities	174
17.11	Controlling marketing activities	174
17.11.1	Outcomes and products of the marketing process	174
17.12	Professional Services Enterprise Public Relations and Management	174
17.12.1	Investigation and forecasting for public relations	175
17.12.2	Planning, estimating and budgeting for enterprise public relations	175
17.12.3	Organising for public relations	175
17.12.4	Activities and processes	175
17.12.5	Leading, coordinating, controlling and communicating public relations activities and processes	175
17.13	Conclusion and recommendations	176
18	Finance, Bookkeeping and Auditing	177
18.1	Introduction to professional service business accounting	177
18.2	Registration and recording processes	177
18.2.1	Project registration	177
18.2.2	Debtor registration	177
18.2.3	Creditor registration	177
18.2.4	Time keeping and recording	177
18.2.5	Disbursement recording and costing rates	179
18.3	Work in process	180
18.3.1	Professional time Work in Process	180
18.3.2	Disbursement Work in Process	180
18.3.3	Work in process management	180
18.4	Professional Services Invoices	181
18.4.1	Definition of an invoice	181
18.4.2	Management of the invoicing cycle	181
18.4.3	Responsibility for issuing of invoices	181
18.5	Professional Service Invoice Specification	181
18.5.1	Value Added Tax (VAT)	182
18.5.2	Accounting records and reporting on invoicing	183
18.5.3	The debtor cycle	183
18.5.4	Credit notes and cancelling of invoices	183
18.5.5	Internal invoicing to personnel	183
18.6	Personnel remuneration and payroll processing	184
18.7	Bookkeeping	184
18.7.1	Accounting general ledger structuring and format	184
18.7.2	The role and use of the general ledger in accounting	184

18.7.3	Accounting system model	187
18.7.4	Accounting software implementation	187
18.8	Processing orders for materials, goods and services	187
18.9	Professional practice finance	188
18.10	Auditing	189
18.11	Conclusion and recommendations	189
19	Personnel Management	191
19.1	Business objects for personnel management	191
19.1.1	High level logical personnel management objects	191
19.1.2	Personnel management objects	191
19.2	Personnel management processes	192
19.3	Payroll management systems	192
19.4	Personnel Debtors	192
19.5	Personnel Creditors	193
19.6	Conclusions and recommendations	193
20	Production	194
20.1	Introduction	194
20.2	Projects and project management	194
20.3	Project management terminology	194
20.4	Production management business objects	194
20.5	Project management processes	196
20.6	Functionality required of a project management system for professional services	196
20.7	Software implementation	197
20.8	Time Management	197
20.9	Projects for administration management	197
20.10	Conclusions and recommendations	197
21	Facilities and Document Management	200
21.1	Introduction to Facilities Management for the Engineering Services Business	200
21.2	Models for facility management	200
21.3	Business objects for facility management	200
21.3.1	Definition of business artefacts and business objects	200
21.4	Management disciplines which relate to business object facility management	201
21.5	Management aspects and business artefacts	201
21.6	Aspects of facilities management activities and processes	201
21.6.1	Operations for office spaces in buildings	201
21.6.2	Monitoring and managing building subsystems	203
21.6.3	Maintenance	203
21.6.4	Risks and exceptional events	203
21.6.5	Health and safety	203
21.6.6	Feedback from the operational environment to the planning and design environment	203
21.7	Software Implementation	203
21.7.1	Asset register	204
21.8	Document management	204
21.8.1	Business objects for document management	204
21.8.2	Document management business processes	204
21.9	Conclusions and recommendations	206
22	Knowledge and Information Management	207
22.1	Introduction to knowledge management	207
22.2	Models for business knowledge management	207
22.2.1	Business objects for knowledge management	207
22.3	Knowledge and information for project execution	208
22.4	Protecting business artefacts against misuse	208
22.5	Conclusions and recommendations	208

23 Logistics	210
23.1 Introduction to logistics for the professional service business enterprise	210
23.2 Models for logistics management	210
23.3 Business objects for logistics management	210
23.4 Professional Service Business Logistics Activities and Process	210
23.4.1 Inbound logistics	211
23.4.2 Outbound logistics	212
23.5 Client Project Logistics	212
23.5.1 Supplier management	212
23.5.2 Materials control	212
23.6 Conclusions and recommendations	212
24 Administration	213
24.1 Introduction to the administrative function	213
24.2 Models for administrative management	213
24.3 Business Objects and Business Administration	213
24.4 Interaction between the administrative processes and other business functions	214
24.4.1 Business Strategy and Policy	214
24.4.2 Marketing Administration	214
24.4.3 Financial Administration	214
24.4.4 Personnel Administration	214
24.4.5 Facilities Administration	214
24.4.6 Logistics Administration Function	215
24.4.7 Project Administration Function	215
24.5 Systems support for administrative management	215
24.6 Administrative Function Reporting Requirements	215
24.7 Conclusions and recommendations	215
25 Risk Management	217
25.1 Introduction to risk management for the professional service business enterprise	217
25.2 Models for risk management	217
25.3 Business objects and risk management	217
25.4 Risk management activities and processes	217
25.4.1 Risk identification	217
25.4.2 Risk analysis and quantification, Risk allocation and control	218
25.4.3 Business function risk analysis, quantification, allocation and control	219
25.4.4 Business infrastructure risk quantification, allocation and control	219
25.4.5 Risk control	219
25.4.6 Risk avoidance and risk reduction	219
25.4.7 Risk financing, retention, transfer and insurance	219
25.5 Financial risk management processes	220
25.5.1 Identification of financial risks	220
25.5.2 Financial risk analysis, quantification, allocation and control	220
25.6 Project Risk Management Processes	221
25.6.1 Project risk identification	222
25.6.2 Project risk analysis and quantification	222
25.6.3 Evolution of Risk Through Project Life Cycle	222
25.6.4 Project Risk Allocation and Insurance	223
25.7 Disaster Recovery Planning	223
25.8 Risk management manual and standard report contents	223
25.9 Conclusions and recommendations	223
26 Practice Management Systems	224
26.1 ProMan by Akron Software	224
26.2 Systems by Deltek Inc.	224
26.3 SAP	225
26.4 Dynamics / Business Solutions / Great Plains by Microsoft	225
26.5 PeopleSoft and JD Edwards by Oracle	226
26.6 Miscellaneous Other Systems	226

27 Conclusions on Addendum	227
V Appendices, Bibliography and References	229
Appendices	230
A MATLAB implementation of set operations	231
B MATLAB implementation of set functions	234
C MATLAB implementation of relation operations	242
C.1 Relation operations programmed in MATLAB	242
D MATLAB implementation of graph operations	244
E MATLAB System Function Example	255
F MATLAB Literal String Processing Functionality	261
G Engineering Process Model	263
G.1 Engineering Process Model Example - MATLAB Code	263
G.2 MATLAB Relational Algebra Functionality	288
G.3 Process model database output MATLAB function	288
G.4 Graph data formats used by the yEd program	294
G.5 Database file transfer format	299
G.6 MATLAB implementation of process model with status settings	300
H Process Model: Person-Task and Person-Data Graphs	314
H.1 Engineering Process Model Graphical Output Example - MATLAB Code	314
H.2 Process model graphical data output MATLAB function	316
I Process Model: Task sequence using data status	318
I.1 Engineering Process Model Example with data status - MATLAB Code	318
J Engineering Process Model Database	348
J.1 Eclipse Development software reference	348
J.2 Eclipse Azzurri Clay Eclipse Plugin for Database Modelling	348
J.3 Azzurri Clay XML DTD Specification file	348
J.4 Database setup SQL statements	350
J.5 PostgreSQL Database Reference	357
J.6 Engineering Process Model - Sample PostgreSQL Database Data Listing	357
J.7 Microsoft Access Database Reference	360
J.8 Access data import process	360
J.8.1 Database schema definition file	360
J.8.2 Access VBA code for data import	363
J.8.3 Database program to import data from database server	365
J.9 PostgreSQL - Importing data into database	366
J.10 Importing database data using the Java JDBC-ODBC bridge	373
J.11 Database application SQL functionality availability and usage	374
J.11.1 SQL Queries for S-Curve presentation	376
J.12 Using Microsoft Data Access Pages	384
K Organisation management and reporting structures using graph applications	394
K.1 Typical Management Reporting Tree Structure	394
K.2 Converting from adjacency matrix format to adjacency list format	397
K.3 Depth first search and tree structure for a given graph	399
K.4 Depth First Search applied to reporting graph structures	402
K.5 Inserting sub management structures into larger structures	409
K.6 Extracting sub graphs linked to selected nodes	413

K.6.1	Graph adjacency matrix to list conversion	417
K.6.2	Graph adjacency list sub graph extraction	418
K.7	Management/ reporting structure - tree analysis examples	419
K.7.1	Basic example	419
K.7.2	Larger more realistic example	426
K.8	Determining connectivity of vertices in graphs e.g. to determine tree vertex links for roll up of reports	440
K.9	Report data roll up using adjacency matrices	441
K.9.1	Theoretical Example using topological sorting and sub matrix extraction	441
K.9.2	Theoretical Example - Using graph adjacency list processing	444
K.9.3	Larger example with numerical values	448
K.9.4	Larger example with string literal values concatenated in accumulation process	461
K.10	Graph sub tree connectivity extraction	479
K.11	Multiple sub tree connectivity extraction	482
K.12	Sub-tree extraction MATLAB functions	484
K.13	File format for yEd graph display program (.tgf)	486
L	Marketing Management - Sample Documents	487
L.1	Marketing activity planning and status reporting	487
L.2	Marketing budgeting and income budget planning	487
L.3	Marketing budget according to project status	487
	Bibliography and References	492

List of Figures

4.1	Uniqueness of R	12
(a)	general	12
(b)	left-unique	12
(c)	right-unique	12
(d)	bi-unique	12
4.2	Quotient set	14
4.3	Canonical mapping	16
4.4	Homogeneous binary relations graph example	20
5.1	Directed graph properties	24
(a)	antireflexive	24
(b)	symmetric	24
(c)	antisymmetric	24
(d)	asymmetric	24
5.2	Graph example	25
5.3	Directed graph example	26
5.4	Simple graph example	26
5.5	Strongly connected components graph	33
5.6	Reduced graph	34
5.7	Strongly connected components	34
5.8	Edge classifications	39
5.9	Graph example	39
5.10	Depth-first-search-forest-example-1	40
5.11	Depth-first-search-forest-example-2	41
6.1	A system with a closed-loop control system	45
6.2	Graphical representation of a formal system structure definition (Pahl [91])	49
6.3	System Function Example	51
7.1	Simplified database system environment logic	62
7.2	Three level schema database architecture	63
7.3	Entity Relationship Diagram Notation	64
7.4	UML Conceptual Schema	64
11.1	Engineering process model relations	91
11.2	Overview of engineering process model binary relations	91
11.3	Overview of engineering process homogeneous binary relations	92
11.4	Rule 1: Order relation in the set of tasks	93
11.5	Rule 2: Order relation in the set of tasks	93
11.6	Rule 3: Order relation in the set of tasks	94
11.7	Task-Dataset relationships: Read, modify, create	95
11.8	Task executed by Person	106
11.9	Task creates dataset	106
11.10	Task reads dataset	106
11.11	Task modifies dataset	106
11.12	Dataset requires tool	107
11.13	Task uses tool	107
11.14	Person executes task	107

11.15	Dataset created by Task	108
11.16	Dataset read by Task	108
11.17	Dataset modified by Task	108
11.18	Tool operates on dataset	108
11.19	Tool operates on dataset	108
11.20	Person operates on dataset	109
11.21	Data operated on by person	109
11.22	Data operated on by person transposed relation from person operates on dataset	109
11.23	Person uses tool	110
11.24	Tool used by person	110
11.25	Data operated on by person	111
11.26	Person operates on dataset via tool	111
11.27	Data operated on by task via tool	112
11.28	Task operates on dataset via tool	112
11.29	Data read modify via task union	112
11.30	Sequence of tasks	113
11.31	Person creates dataset	113
11.32	Data read modify via person union	113
11.33	Person Loading	114
11.34	Tool Loading - tools required with other tools	114
11.35	Dataset history with dataset determined via persons - read only	115
11.36	Person data read and modify via data union	115
11.37	Dataset history with dataset determined via persons - read and modify	116
11.38	Dataset history with dataset determined via tasks - read only	117
11.39	Task reads and modify via data union	117
11.40	Dataset history with dataset determined via tasks - read and modify	118
11.41	Tasks and data for client	119
11.42	Tasks and data for architect	119
11.43	Tasks and data for structural engineer	119
11.44	Tasks and data for technologist	119
11.45	Tasks and data for checking engineer	119
11.46	Person executes task relation	120
11.47	Tasks creates data relation	120
11.48	Tasks reads data relation	121
11.49	Tasks modifies data relation	121
11.50	Data requires tool relation	122
11.51	Person requires tool relation	122
11.52	Person reads data relation	123
11.53	Tasks requires tool relation	123
11.54	Task sequence - Rule 1	124
11.55	Task sequence - Rule 2	124
11.56	Task sequence - Rules 1 & 2	125
11.57	Task sequence - Rules 1, 2 & 3	125
11.58	Step schedule of Tasks	126
12.1	Reporting structure graph representation - level 1 - from adjacency matrix	127
12.2	Reporting structure graph representation - level 2 - from adjacency matrix	128
12.3	Reporting structure graph representation - all levels - from adjacency matrix	129
12.4	Reporting structure adjacency matrix	130
12.5	Reporting structure adjacency matrix	131
12.6	Reporting structure adjacency matrix	131
12.7	Reporting structure adjacency matrix - Level 1 to Level 0	132
12.8	Reporting structure adjacency matrix - Level2 to Level 1	132
12.9	Task step schedule adjacency matrix	132
12.10	Topological sorting of the task step schedule	135
12.11	Sub-adjacency matrix for logical step 1	136
12.12	Sub-adjacency matrix for logical step 2	136
12.13	Sub-adjacency matrix for logical step 3	137

12.14	Sub-adjacency matrix for logical step 4	138
12.15	Sub-adjacency matrix for logical step 5	138
13.1	Database demonstration UMLDeploymentDiagram	140
13.2	Engineering Process Model - Clay program screen	141
13.3	Process model database - DBVisualizer summary display	143
13.4	Process model database table print - DBVisualizer	144
13.5	DBVisualizer extended display of process model database	145
13.6	Access database structure report	146
13.7	Microsoft Windows Datalink tabbed panes	147
	(a) Connection	147
	(b) Provider	147
	(c) Advanced	147
	(d) All	147
13.8	Access DLL library reference display	147
13.9	Engineering Process Example - Access Report	148
13.10	Engineering Process Example - Excel tabular report	149
13.11	Engineering Process Example - Excel chart	150
13.12	PLEP Java Application Object Structure gif	150
13.13	Microsoft Access Output Field Specification	151
	(a) General field properties	151
	(b) Query definition pane	151
13.14	Access database tabular reporting	152
13.15	Database tabular reporting with browser data access pages	153
13.16	Database reporting with browser data access pages	154
15.1	Systems Theory Based Professional Engineering Services Enterprise Management Information Systems	160
16.1	General Management Aspects according to De Villiers	164
17.1	The nature and roles of service marketing according to Leonard Berry	170
18.1	Basic time sheet example	180
18.2	Tax Invoice Sample Screen	182
18.3	Engineering Services Enterprise Systems	190
20.1	Salford Process Protocol - Part A	198
20.2	Salford Process Protocol - Part B	199
25.1	Project risk management model	221
25.2	Project risk management model	222
26.1	PLEP Engineering Process Software	226
J.1	EngineeringProcessModelClay	392
J.2	Altova XMLSPy splash display	393
K.1	Reporting or management structure - graph representation - from adjacency matrix: (a) Reporting structure graph representation - level 1; (b) Reporting structure graph representation - level 2; (c) Reporting structure graph representation - all levels;	398
	(a) Graph - level 1	398
	(b) Graph - level 2	398
	(c) Graph - all levels	398
K.2	Reporting or management structure - DFS tree representation: (a) Reporting structure DFS tree representation - level 1; (b) Reporting structure DFS tree representation - level 2; (c) Reporting structure DFS tree representation - all levels;	410
	(a) DFS tree - level 1	410
	(b) DFS tree - level 2	410
	(c) DFS tree - all levels	410

K.3	Extracting sub-trees linked to vertices management structure - only non-null entries shown	
	(a) Reporting structure tree representation (b) Sub-tree representation - vertex 1 - M_0 ; (c) Sub-tree representation - vertex 2 - M_A ; (d) Sub-tree representation - vertex 3 - M_B ;	417
	(a) Reporting structure tree representation	417
	(b) Sub-tree - vertex 1 - M_0	417
	(c) Sub-tree - vertex 2 - M_A	417
	(d) Sub-tree - vertex 3 - M_B	417
K.4	Reporting structure adjacency matrix	420
K.5	Reporting structure adjacency matrix	425
K.6	Reporting structure adjacency matrix - Level 1 to Level 0	426
K.7	Reporting structure adjacency matrix - Level2 to Level 1	426
K.8	Reporting structure adjacency matrix	440
K.9	Sub-tree connectivity extraction per vertex as listed: (a) Example tree graph; (b) Vertex a - empty graph; (c) Vertex b; (d) Vertex c; (e) Vertex d; (f) Vertex e; (g) Vertex f; (h) Vertex g;	483
	(a) Example tree graph	483
	(b) Vertex a - empty graph	483
	(c) Vertex b	483
	(d) Vertex c	483
	(e) Vertex d	483
	(f) Vertex e	483
	(g) Vertex f	483
	(h) Vertex g	483
K.10	Sub-tree extraction	484
	(a) Example tree graph	484
	(b) Multiple sub-trees extracted for vertices a,e and g	484
	(c) Multiple sub-trees extracted for vertices b and g	484
L.1	GFJ Inc - Marketing activity planning sheet	488
L.2	GFJ Inc. - Sample marketing management action status report	489
L.3	GFJ Inc. - Sample marketing management budget derived from control report	490
L.4	Puttergill -Sample corporate budget showing FEE income as per project marketing classification	491

List of Tables

4.1	Properties of relations	12
5.1	Properties of directed graphs	23
5.2	Strongly connected components	34
5.3	Pre-order and post-order numbers for example 1	40
5.4	Pre-order and post-order numbers of example 2	40
6.1	System classification according to Kenneth Boulding [19]	47
6.2	Systems classified according to mode of operation and the physical nature of their components and couplings; Jones and Edited by Singleton, W.T. et al. [66]	48
6.3	System set theoretic concept	49
6.4	Set theoretic system definition	50
6.5	System function formal specification	51
6.6	System function example	52
6.7	Classification hierarchies used in business organisation	59
7.1	General format of a relation represented as a table	66
7.2	Some standard SQL data types	72
7.3	Operations of Database Relational Algebra	76
7.4	Comparison of Relational Algebra and Calculi	79
7.5	The PROJECT Table with duplicate entries	80
7.6	The PROJECT Table	80
7.7	The PROJECT-EMPLOYEE Table	81
9.1	MATLAB standard logical operations and functions on boolean (logical) variables used . . .	87
9.2	MATLAB - Basic Relational Algebra Operations using Boolean matrix representation of relations	88
9.3	Zero, One and Identity boolean matrices	88
11.1	Engineering process model concept	98
11.2	Relations specified for model example in consulting engineering process	98
11.3	Boolean adjacency matrix representation of relation Persons executes Task $\mathbf{R}_{Person-Task}$.	98
11.4	Boolean adjacency matrix representation of relation Task creates Dataset $\mathbf{R}_{Task-DataCreate}$	98
11.5	Boolean adjacency matrix representation of relation Task reads Dataset $\mathbf{R}_{Task-DataRead}$. .	99
11.6	Boolean adjacency matrix representation of relation Task modifies Dataset $\mathbf{R}_{Task-DataModify}$	99
11.7	Boolean adjacency matrix representation of relation Dataset operated on by Tool $\mathbf{R}_{Data-Tool}$	99
11.8	Boolean adjacency matrix representation of relation Task uses Tool $\mathbf{R}_{Task-Tool}$	99
11.9	Data status values	103
12.1	Management reporting relation	127
12.2	Hihger level management reporting relation	128
12.3	Complete multi level management reporting relation	128
12.4	Management reporting graph levels	131
12.5	Tasks for graph structure analysis	133
16.1	Maintaining a strong and healthy strategy - Robert [102]	162
17.1	Differences between services and product marketing	169

18.1	Finance, accounting and bookkeeping processes and objects	178
18.2	Finance, accounting and bookkeeping processes and objects (continued)	179
18.3	Professional service invoice specification	182
18.4	Information hierarchies which link to invoice specification and generation	182
18.5	General Ledger Basic Sales Accounts	183
18.6	General Ledger VAT Accounts	183
18.7	Personnel and payroll general ledger accounts	184
18.8	General ledger matrix model	185
18.9	Debit and credit transaction logic: effect on account balances	185
18.10	Overview of the typical structure of a general ledger - Assets and Liabilities	187
18.11	Overview of the typical structure of a general ledger - Income / Expenses	188
20.1	Project management concepts and terms	195
21.1	Facility Management Functions	202
21.2	Relation between business objects and management disciplines	202
21.3	Engineers and Facility- and Practice Infrastructure Management	202
21.4	Management activities relating to business artefact classification	205
21.5	Document media formats	205
22.1	Business knowledge management objects	207
23.1	Business objects requiring logistics management	210
23.2	Project document classification	211
24.1	Disposal and Retention of Business Documents	216
24.2	South African Business Legal Forms	216
K.1	Management reporting graph levels	425
K.2	Data file for tasks : PEPEXTsequenceT.tgf	437

Nomenclature

Sets

\in	Element of, or contained in
\notin	Not an element of, not contained in
\Leftrightarrow	If and only if
\wedge	Logical or
\vee	Logical and
$/$	Logical negation
U	Universal set or universe of discourse
\bigwedge	Universal qualifier - for every
\forall	Universal qualifier - for every
\bigvee	Existential qualifier - there exists or is
\exists	Existential qualifier - there exists or is
ϕ	Empty set
\subset	Proper subset
\subseteq	Subset
\cup	Union of sets
\cap	Intersection of sets
$-$	Set difference
\overline{A}	Complement of set A
\mathbb{Z}	Integer numbers
\mathcal{N}	Natural numbers
\mathcal{Q}	Rational numbers
\mathbb{R}	Real numbers

Relations

(a, b)	Ordered pair
$A \times B$	Cartesian product of sets A and B
R	Relation
I_A	Identity relation
R^{-1}	Inverse or dual relation
$R \circ S$	Composition of relations R and S
$\Phi : A \rightarrow Z$	Mapping from A to Z
$card(A) = A $	Cardinality
ϕ	Null relation
I	Identity relation
E	All relation

Graph Theory

$G = (V; E)$	Graph
--------------	-------

G	Graph
V	Vertex set of graph
E, R	Edge set of graph
u, v, x, y, x_1, y_1	Graph vertices
$e = \{u, v\}$	Graph edge
\mathbf{R}, \mathbf{A}	Graph adjacency matrix
\mathbf{B}	Graph incidence matrix
(x, y)	Graph vertex pair
$r(x)$	Rank of a graph vertex
Φ	Vertex set mapping
G_K	Reduced graph
V_K	Reduced vertex set
R_K	Reduced edge set

System Theory

C	Set of internal system elements
B	Set of system boundary elements
E	Set of system environment elements
I_s	Set of system inputs
O_s	Set of system outputs or readouts
S_s	Set of system states
N_s	System next state mapping
T	System time or progress counters
f	System trajectory

Database Theory

R	Relational schema
H	Relation header
B	Relation body
K	Key of a relation
$r()$	Relation state
$dom(A_i)$	Attribute domain
$ D $	Domain cardinality
A_i	Attribute
\forall	Universal qualifier - for every
\exists	Existential qualifier - there exists or is
σ	Selection operation
π	Projection operation
\sqcap	Tuple concatenation for tuples s and t
\cup	Set union operation
$-$	Set difference operation
\cap	Set intersection operation
\otimes	Set Cartesian product operation
\bowtie	General join operation
\leftarrow	Natural join operation
$\sqcup\bowtie$	Left outer join
$\bowtie\sqcup$	Right outer join
$\sqcup\bowtie\sqcup$	Full outer join

Chapter 1

Introduction

The primary goal of this thesis is to indicate how systems theory and engineering process modelling can be applied to provide models for consulting engineering service business enterprises. The typical management systems used for these businesses are investigated to determine the application of systems and process models.

The motivation for this study is based on the fact that integrated management systems for consulting engineering practices are presently based on selective business analysis and process modelling that has evolved over time, as reported in a survey and study by Smit [110]. Furthermore, current engineering management systems are simply computer implementations of management procedures based on techniques that were developed to solve problems in the absence of the computational capabilities of the modern computer. To rectify this, a fundamental approach to analyse the business and management functions using systems theory and engineering process modelling techniques is required, which has not been attempted to date. This study develops and demonstrates the application of fundamental analysis in consulting engineering enterprise management and reviews advantages that can be obtained from using this approach.

It is shown that the mathematical Algebra of Relations and associated Graph Theory provide the mathematical basis on which management problems can be treated systematically. Since these fields of mathematics are well developed and very broad, the essential parts of the theories are identified. Thereupon, the application of the very abstract mathematical concepts to two important and typical engineering management problems are developed, which represents the core contribution of the dissertation.

The study is developed in two parts and an addendum:

1. This part provides an overview of the necessary mathematical theory required to support development of business models.
2. Management systems theory and relation based engineering process modelling techniques are applied in this part to build generic enterprise models and data processing models. These models provide inputs for the management processes of professional service business enterprises. The outcome of the modelling and analysis is a set of database models with reporting functionality, to be used in the management process. A demonstration of technology available for development of the models and techniques, described in the previous part, is undertaken in this part. Generic implementations of database models and reporting techniques for systems which deal with management data in a consulting engineering business are described and demonstrated.
3. In the addendum techniques and technology developed in the previous parts are used to identify typical models and system functionality needed to support the management functions of the consulting engineering service business.

To limit the scope of the study it was decided to focus on the use of an engineering process model approach for project production planning and management. Relational models for the engineering process are developed and a database implementation is done for the process model.

Techniques to process management reports referring to business organisational structures are developed and demonstrated. These techniques are based on a graph theoretical approach.

The appendices which contain the computer program implementations of the various examples discussed in the document are available on the CD disk included with the document.

Part I

Management systems theory and engineering process modelling techniques

Chapter 2

Overview of Part I

This part deals with management systems theory and engineering process modelling techniques. Management systems theory and engineering process modelling techniques are applied to build enterprise models and data processing models which provide inputs for the management processes of the business enterprise. The theoretical models are based on a mathematical approach using set theory, graph theory and relational algebra. The outcome of the modelling and analysis are a set of database models with reporting functionality, to be used in the management process.

The primary goal is to study systems theory and engineering process modelling techniques which can be applied to models of consulting engineering service business enterprises and the typical management systems used for these businesses.

In this part the basic mathematical theory and system theory required to discuss, analyse and design management system models are described.

The applicable mathematical and systems theory dealt with is:

- Basic set theory
- Relations and mappings based on set theory
- Graph theory
- General systems theory based on set theory
- Database theory

Set theory, theory of relations and graph theory are not treated in current engineering curricula. Therefore a review of these theories which form the basis of discrete mathematics are included in this document.

Examples pertaining to management of consulting business enterprises are described and MATLAB code developed to support the demonstration of the theoretical concepts involved is included in appendices to the document.

Conclusion, recommendations and suggestions for further work

The theory set out in this part is applied in part II to develop technology which can be used in engineering service enterprise management systems.

Chapters 3, 4, 5, 6 and 7 will be of value to students and researchers working in the field of discrete mathematics applied to engineering.

The MATLAB functionality developed for this study to implement the theory described in chapters 3, 4, 5 and 6 can be of value in the teaching of concepts in this field of study.

Chapter 3

Set Theory

3.1 Introduction

The basic concepts and terminology of set theory which may be applied to business systems modelling are reviewed in this chapter. The development follows directly from that by Pahl and Damrath [92], Cronje [26] and Lipschutz [73, 74]. Selected paragraphs taken from these references are used as such.

3.2 Sets, Elements of Sets and Subsets

This section gives a basic definition of a set. The specification of elements of a set as well as families of elements of sets using subscripts and the use of quantifiers for set elements is defined. The concepts of equality of sets, subsets, sets of sets, power set and set comparability are then dealt with.

3.2.1 Definition of a set

A collection of elements with similar well-defined properties is called a set. The adjective well-defined is used to emphasise the basic requirement that one must always be able to specify a set that, given any object whatsoever one must be able to determine if the object belongs to the set in question or not. Objects which are separable and can be uniquely identified are called elements. Each property of an element of a set is described either by its value or by rules for determining its value. The elements of a set are uniquely identified using a property of the elements which takes different values for all elements of the set. This property is called the name (label, identifier) of the element.

3.2.2 Defining elements of sets

A set M is specified either by enumerating the designations of the elements or by describing the properties of the elements. The order of enumeration of the elements is irrelevant. If two elements in the enumeration bear the same designation, they represent the same element. Each element is contained in the set only once. The set without elements is called the empty set and is designated by ϕ .

$$\begin{array}{lll} M & = & \{a, b, c\} \quad \text{set } M \text{ consists of the elements } a, b, c \\ M & = & \{x \mid E(x)\} \quad \text{set } M \text{ contains every element for which the logical expression } E(x) \text{ is true} \\ \phi & := & \{x \mid x \neq x\} \quad \text{empty set} \end{array} \tag{3.2.1}$$

The membership of an element a in a set M is represented using the symbols \in and \notin :

$$\begin{array}{ll} a \in M & a \text{ is an element of } M \\ a \notin M & a \text{ is not an element of } M \end{array}$$

Sets can be finite or infinite. A set is finite if the counting process of its elements comes to an end. Otherwise the set is infinite. Refer to the section on cardinality 4.15.1 where the concept of finite and infinite sets are defined without reference to the natural numbers.

3.2.3 Defining families of elements

Designating the elements of a set by different names is inconvenient for sets with a large number of elements. Therefore the elements of a set X are often designated by x_1, x_2, x_3, \dots . The common designation by the lowercase letter x symbolises membership in the set X , while the index $i \in \{1, 2, 3, \dots\}$ identifies the element. The elements x_i are called a family of elements. The family of elements is designated by $\{x_i\}$.

$$X = \{x_i \mid i \in I = \{1, 2, 3, \dots\}\} \quad (3.2.2)$$

3.2.4 Universal and existential quantifiers

There are statements which are true for certain elements of a set M and false for other elements of M . Using the universal quantifier \bigwedge and the existential quantifier \bigvee one has:

$$\begin{aligned} \bigwedge_{x \in M} a(x) & \text{ for every } x \text{ in the set } M, a(x) \text{ holds} \\ \bigvee_{x \in M} a(x) & \text{ there is an } x \text{ in the set } M \text{ for which } a(x) \text{ holds} \end{aligned} \quad (3.2.3)$$

3.2.5 Equality of sets

Two sets A and B are said to be equal if they contain the same elements. If the sets A and B are equal, they contain the same elements. The statement $A = B$ (A equals B) can either be true or false.

$$(A = B) :\Leftrightarrow \bigwedge_x (x \in A \Leftrightarrow x \in B) \quad (3.2.4)$$

$$\begin{aligned} A = B & \text{ sets } A \text{ and } B \text{ are equal} \\ A \neq B & \text{ sets } A \text{ and } B \text{ are not equal} \end{aligned}$$

3.2.6 Subsets of a set

Set A is called a subset of a set B if every element of A is also an element of B . If the set B contains at least one element not contained in A , then A is called a proper subset of B .

$$\begin{aligned} (A \subseteq B) & :\Leftrightarrow \bigwedge_x (x \in A \Rightarrow x \in B) \\ (A \subset B) & :\Leftrightarrow \bigwedge_x (x \in A \Rightarrow x \in B) \wedge \neg(A = B) \end{aligned} \quad (3.2.5)$$

In addition to the symbols \subseteq (contained in) and \subset (properly contained in), the symbols \supseteq (includes) and \supset (properly includes) are also used.

$$\begin{aligned} B \supseteq A & \text{ set } B \text{ includes set } A & A \subseteq B & A \text{ is a subset of } B \\ B \supset A & \text{ set } B \text{ properly includes set } A & A \subset B & A \text{ is a proper subset of } B \end{aligned}$$

3.2.7 Comparable sets

Sets A and B are said to be comparable if $A \subset B$ or $B \subset A$. One of the sets is a subset of the other. Sets are not comparable if $A \not\subset B$ and $B \not\subset A$. For equal sets A and B $A \subset B$ and $B \subset A$ holds at the same time.

3.2.8 Sets of sets

The elements of a set can be sets themselves. The terms a ‘family of sets’ or ‘class of sets’ can be used to denote a ‘set of sets’.

3.2.9 Power set

The power set is an example of a set of sets. From a given set M of n elements, 2^n subsets can be formed, including ϕ and M . The set of all subsets of M , including ϕ and M , is called the power set of M and is designated by $P(M)$. The set M is called the reference set of the power set $P(M)$.

$$\begin{aligned}
M &= \{a, b, c\} & n = 3, \quad 2^3 = 8 \\
P(M) &= \{\phi, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}
\end{aligned}
\tag{3.2.6}$$

3.2.10 Universal set

In a given set theoretic application all the sets under investigation will typically be subsets of a given fixed set. This set is called the universal set U . U is also known as the universe of discourse for the set application at hand.

3.2.11 Disjoint sets

If sets A and B have no elements in common i.e. no element of A is contained in B and no element of B is contained in A the sets A and B are disjoint.

$$A \text{ and } B \text{ disjoint} \Rightarrow A \cap B = \phi \tag{3.2.7}$$

3.2.12 Partition of a set

If S is a non-empty set, a partition of S is a subdivision of S into non-overlapping non-empty sets. A partition of a set S is a collection of sets (set of sets) $\{A_i\}$ where:

$$\begin{aligned}
&\forall i \ A_i \neq \phi \\
&\forall a \in S \ \exists i \ni a \in A_i && \text{every } a \in S \text{ belongs to one of the } A_i \\
&A_i \neq A_j \Rightarrow A_i \cap A_j = \phi && \text{the sets in } \{A_i\} \text{ are mutually disjoint}
\end{aligned}
\tag{3.2.8}$$

3.3 Set Operations

In this section the binary set operations, i.e. set union, set intersection and set difference and the unary operation, set complement are defined.

3.3.1 Union of sets

The union $A \cup B$ of set A and set B is the set of all elements belonging to set A or set B or set A and set B .

$$\begin{aligned}
A \cup B &:= \{x \mid x \in A \vee x \in B\} \\
A \cup B &= B \cup A
\end{aligned}
\tag{3.3.1}$$

3.3.2 Intersection of sets

The intersection $A \cap B$ of set A and set B is the set elements which are common to both sets A and B , i.e. belonging to both set A and set B .

$$\begin{aligned}
A \cap B &:= \{x \mid x \in A \wedge x \in B\} \\
A \cap B &= B \cap A
\end{aligned}
\tag{3.3.2}$$

3.3.3 Set difference

The difference of set A and set B , $A - B$ is the set of elements belonging to set A but not to set B . This is read as A difference B or as A minus B .

$$\begin{aligned}
A - B &:= \{x \mid x \in A \wedge x \notin B\} \\
A - B &\subset A
\end{aligned}
\tag{3.3.3}$$

3.3.4 Complement of a set

If a set A is the subset of a set M then the difference of A with respect to M is defined as $M - A$ and is designated as \overline{A} .

$$\begin{aligned}
\bar{A} &:= M - A \\
A \cup \bar{A} &= M \\
A \cap \bar{A} &= \phi
\end{aligned}
\tag{3.3.4}$$

3.4 Sets of Numbers

Basic sets of numbers and intervals on these sets of numbers which are used in business systems and systems models are introduced in this section.

3.4.1 Integer numbers

The set of integer numbers \mathcal{Z} (or integers) contains the positive and negative whole numbers including zero.

$$\mathcal{Z} := \{\dots, -2, -1, 0, 1, 2, 3, \dots\} \tag{3.4.1}$$

3.4.2 Natural numbers

The set of natural numbers \mathcal{N} contains all the positive integers.

$$\mathcal{N} := \{1, 2, 3, \dots\} \tag{3.4.2}$$

3.4.3 Rational and irrational numbers

The set of rational numbers \mathcal{Q} contains numbers which can be expressed as the ratio of two integers.

$$\mathcal{Q} := \left\{ x \mid x = \frac{p}{q}, p \in \mathcal{Z} \text{ and } q \in \mathcal{Z} \right\} \tag{3.4.3}$$

The set of irrational numbers \mathcal{Q}' are those numbers which cannot be expressed as the ratio of two integers. Examples of irrational numbers are $\sqrt{2}$, $\sqrt{3}$, π and e .

3.4.4 Real numbers

The set of real numbers \mathfrak{R} can be defined as the union of the sets of rational (\mathcal{Q}) and irrational (\mathcal{Q}') numbers. The set of real numbers and its properties are called the real number system. One of the most important properties of the real numbers is that they can be represented by the points on a straight line – the real line. Every real number can be represented in nonterminating decimal format. The rational numbers correspond to those decimals where the block of digits (zero included) is continually repeated, while the irrational numbers correspond to the other nonterminating decimals. The symbol \mathfrak{R} is used for the real numbers to distinguish the set of real numbers from the symbol R used for a relation later.

$$\begin{aligned}
3/8 &= .375 \\
3/8 &= .375000000\dots \\
3/8 &= .374999999\dots \\
2/11 &= .1818181\dots \\
\pi &= 3.1417\dots
\end{aligned}
\tag{3.4.4}$$

3.4.5 Number inequalities

The ordering of the real numbers \mathfrak{R} is achieved by defining the less than relation $<$.

$$\begin{aligned}
a < b &: b - a \text{ a positive number} \\
b < a &: a - b \text{ a positive number}
\end{aligned}
\tag{3.4.5}$$

The following properties of the relation $<$ can be proved.

$$\begin{aligned}
 &\text{Either } a < b, a = b \text{ or } b < a \\
 &\text{If } a < b \text{ and } b < c \text{ then } a < c \\
 &\text{If } a < b \text{ then } a + c < b + c \\
 &\text{If } a < b \text{ and } c \text{ is positive then } ac < bc \\
 &\text{If } a < b \text{ and } c \text{ is negative then } bc < ac
 \end{aligned}
 \tag{3.4.6}$$

If $a < b$ then the point on the real line representing a lies to the left of the point representing b .

3.4.6 Absolute value

The absolute value of a real number x is denoted by $|x|$ is defined as:

$$|x| = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{if } x < 0 \end{cases}
 \tag{3.4.7}$$

3.4.7 Intervals on sets of numbers

Finite and infinite intervals on ordered sets such as sets of numbers can be defined. In the case of the real numbers the order relation is $<$ applied to form the intervals.

Finite intervals which are defined on ordered sets can be defined for the set of real numbers in one of the following ways:

$$\begin{aligned}
 [a, b] &:= \{x \in \mathbb{R} \mid a \leq x \leq b\} \text{ a closed interval} \\
 (a, b) &:= \{x \in \mathbb{R} \mid a < x < b\} \text{ an open interval} \\
 [a, b) &:= \{x \in \mathbb{R} \mid a \leq x < b\} \text{ a left closed right open interval} \\
 (a, b] &:= \{x \in \mathbb{R} \mid a < x \leq b\} \text{ a left open right closed interval}
 \end{aligned}
 \tag{3.4.8}$$

Defining \mathcal{I} to be the family of all intervals on the real line containing intervals A and B , the following properties of intervals can be stated:

- The intersection of two intervals is an interval.

$$A \in \mathcal{I}, B \in \mathcal{I} \Rightarrow A \cap B \in \mathcal{I}.
 \tag{3.4.9}$$

- The union of two non-disjoint intervals is an interval.

$$A \in \mathcal{I}, B \in \mathcal{I}, A \cap B \neq \emptyset \Rightarrow A \cup B \in \mathcal{I}.
 \tag{3.4.10}$$

- The difference of two non-comparable intervals is an interval.

$$A \in \mathcal{I}, B \in \mathcal{I}, A \not\subset B, B \not\subset A \Rightarrow A - B \in \mathcal{I}.
 \tag{3.4.11}$$

Infinite intervals can be formed as:

$$\begin{aligned}
 [a, \infty) &:= \{x \in \mathbb{R} \mid x \geq a\} \text{ a left closed infinite interval} \\
 (-\infty, \infty) &:= \{x \in \mathbb{R}\} \text{ an open interval - all the real numbers} \\
 (-\infty, b] &:= \{x \in \mathbb{R} \mid x \leq b\} \text{ a left infinite right closed interval}
 \end{aligned}
 \tag{3.4.12}$$

3.4.8 Bounded and unbounded sets of numbers

If A is a set of numbers then A is a bounded set if A is the subset of a finite interval.

Set A is bounded if:

$$\exists M > 0 \text{ such that } |x| \leq M \forall x \in A
 \tag{3.4.13}$$

A set is unbounded if it is not bounded. If a set is finite it is necessarily bounded. If a set is infinite it can be bounded or unbounded.

3.5 MATLAB implementation of Set Operations

Example MATLAB function implementations of selected set concepts, definitions and operations are included in Appendix A and Appendix B. These functions are used later in part II for business system examples.

Chapter 4

Relations and Mappings

4.1 Introduction

The basic concepts and terminology of relations based on set theory which may be applied to business systems modelling are reviewed in this chapter. The development follows directly from that by Pahl and Damrath [92], Cronje [26], Lipschutz [73, 74] and Open University Mathematics Foundation Course Team [89]. Selected paragraphs taken from these references are used as such.

4.2 Ordered pair

In a set, the order of elements is irrelevant, so that $\{a, b\} = \{b, a\}$. Two elements a and b whose order is relevant are called an ordered pair. An ordered pair is enclosed in parentheses. The elements a and b may be contained in different sets.

Two ordered pairs (a, b) and (c, d) are equal if and only if $a = c$ and $b = d$.

$$\begin{array}{ll} \text{ordered pair} & (a, b) := \{\{a\}, \{a, b\}\} \\ a & \text{first component of the ordered pair } (a, b) \\ b & \text{second component of the ordered pair } (a, b) \end{array} \quad (4.2.1)$$

4.3 Cartesian product

Given sets A and B , the set of all ordered pairs (a, b) that can be formed using elements $a \in A$ and $b \in B$ is called the Cartesian product (direct product) of the sets A and B . The Cartesian product is designated by $A \times B$. The Cartesian product of a set by itself can also be formed and is written as A^2

$$\begin{aligned} A \times B &:= \{(a, b) \mid a \in A \wedge b \in B\} \\ A \times A &:= \{(a, b) \mid a \in A \wedge b \in A\} \end{aligned} \quad (4.3.1)$$

Multiple Cartesian products can also be formed as shown below.

$$\begin{aligned} A \times B \times C &:= \{((a, b), c) \mid a \in A \wedge b \in B \wedge c \in C\} \\ A \times A \times A &:= \{((a, b), c) \mid a \in A \wedge b \in A \wedge c \in A\} \end{aligned} \quad (4.3.2)$$

The order of the formation of a multiple Cartesian product is important seeing that the Cartesian product is not distributive over itself.

4.4 Unary relations

A unary relation forms a subset of a set. Let a non-empty set M of elements and a unary operation on these elements be given. The value of the unary operation Ra for an element a is true or false.

$$\begin{aligned} u &:= \{a \in M \mid Ra\} \subseteq M \\ [u &\subseteq M] \end{aligned} \quad (4.4.1)$$

4.5 Binary relations

A relation on two sets is called a binary relation. A binary relation is a set of ordered pairs of elements. It is a subset of a Cartesian product of two sets. A relation on two sets, or a heterogeneous binary relation, is a subset of the Cartesian product of two different sets.

A relation in a set A , or a homogeneous binary relation, is a subset of the Cartesian product $A^2 = A \times A$, i.e. where the two factors of the product coincide.

4.6 Heterogeneous binary relation

Given two non-empty sets A and B , with a binary operation for a relation R on the elements $a \in A$ and $b \in B$ whose value is a logical constant. The value of the operation for the ordered pair (a, b) in the product $A \times B$ is designated by aRb (a is related to b) and is either true or false.

This implies that if A and B are sets, a binary relation R assigns to each ordered pair (a, b) contained in $A \times B$ exactly one of the statements given in equation 4.6.1.

$$\begin{aligned} &\text{'a is related to b', written as } aRb \\ &\text{'a is not related to b', written as } a \neg Rb \end{aligned} \quad (4.6.1)$$

The subset R of pairs (a, b) for which aRb is true is called a relation on A and B , or a heterogeneous binary relation. Thus the relation is a set containing the ordered pairs of elements for which the relationship specified by the operation holds. The order of the elements a and b in the operation is relevant to the result of the operation. The relation R is a subset of the heterogeneous Cartesian product $A \times B$.

$$R := \{(a, b) \in A \times B \mid aRb\} \subseteq A \times B \quad (4.6.2)$$

4.7 Homogeneous binary relation

Consider a non-empty set M , with a binary operation for a relation R on the elements $a \in M$ and $b \in M$ whose value is a logical constant. The value of the operation for the ordered pair (a, b) in the product $A \times A$ is designated by aRb and is either true or false.

The subset R of pairs (a, b) for which aRb is true is called a relation in M , or a homogeneous binary relation. Thus the relation is a set containing pairs of elements for which the relationship specified by the operation holds. The corresponding homogeneous relation is the set of all ordered pairs (a, b) for which the binary operation aRb is true. It is a subset of the homogeneous Cartesian product $M \times M$.

$$R := \{(a, b) \in M \times M \mid aRb\} \subseteq M \times M \quad (4.7.1)$$

4.8 Properties of relations

The subset $R \subseteq M \times M$ of the Cartesian product of a set with itself for which aRb is true is called a relation in M . The relationships between the statement values aRb and bRa of the pairs (a, b) and (b, a) determine the properties of the relation. These properties are defined in Table 4.1 for $a, b, c \in M$.

4.9 Totality of a relation on A and B

The subset $R \subseteq A \times B$ for which aRb is true is a relation on the sets A and B . The subset of A for which there exists $b \in B$ such that aRb is true is called the domain of R . The subset of B for which there exists $a \in A$ such that aRb is true is called the codomain of R . The relation is said to be left-total if its domain is A . The relation is said to be right-total if its range is B . A relation which is left- and right-total is said to be bitotal.

$$\begin{aligned} R \text{ is left-total} &: \Leftrightarrow \bigwedge_a \bigvee_b (aRb) \\ R \text{ is right-total} &: \Leftrightarrow \bigwedge_b \bigvee_a (aRb) \\ R \text{ is bitotal} &: \Leftrightarrow R \text{ is left-total} \wedge R \text{ is right-total} \end{aligned} \quad (4.9.1)$$

R is reflexive	$:\Leftrightarrow \bigwedge (aRa)$
R is antireflexive	$:\Leftrightarrow \bigwedge_a (\neg aRa)$
R is symmetric	$:\Leftrightarrow \bigwedge_a \bigwedge_b (aRb \Rightarrow bRa)$
R is asymmetric	$:\Leftrightarrow \bigwedge_a \bigwedge_b (aRb \Rightarrow \neg bRa)$
R is antisymmetric	$:\Leftrightarrow \bigwedge_a \bigwedge_b (aRb \wedge bRa \Rightarrow a = b)$
R is linear	$:\Leftrightarrow \bigwedge_a \bigwedge_b (aRb \vee bRa)$
R is connex	$:\Leftrightarrow \bigwedge_a \bigwedge_b (a \neq b \Rightarrow aRb \vee bRa)$
R is transitive	$:\Leftrightarrow \bigwedge_a \bigwedge_b \bigwedge_c (aRb \wedge bRc \Rightarrow aRc)$

Table 4.1: Properties of relations

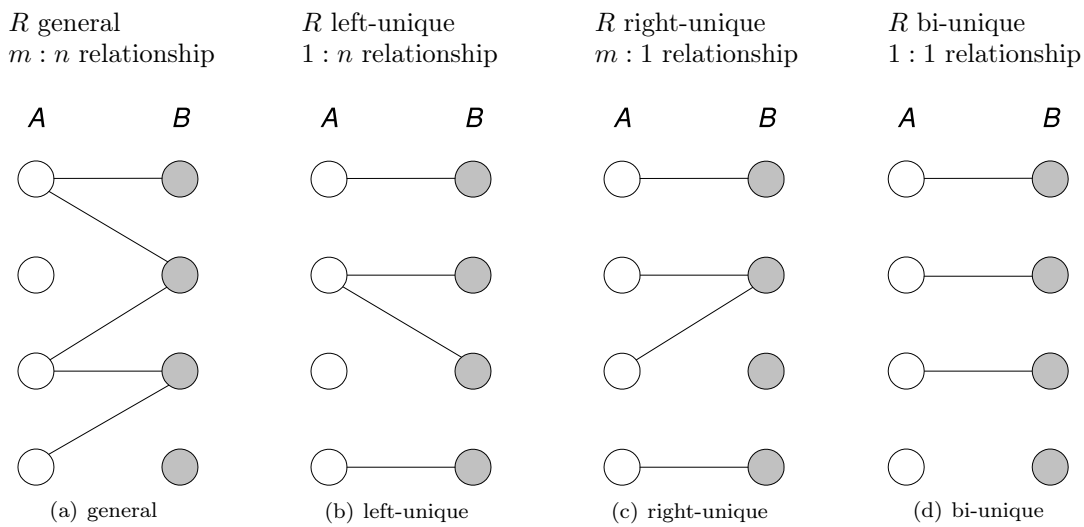
4.10 Uniqueness of a relation on A and B

A relation on A and B is said to be left-unique if the statements aRb and cRb are true only for $a = c$. The relation is said to be right-unique if the statements aRb and aRc are true only for $b = c$. A relation which is left-unique and right-unique is said to be bi-unique.

$$\begin{aligned}
 R \text{ is left-unique} &:\Leftrightarrow \bigwedge_a \bigwedge_b \bigwedge_c (aRb \wedge cRb \Rightarrow a = c) \\
 R \text{ is right-unique} &:\Leftrightarrow \bigwedge_a \bigwedge_b \bigwedge_c (aRb \wedge aRc \Rightarrow b = c) \\
 R \text{ is bi-unique} &:\Leftrightarrow R \text{ is left-unique} \wedge R \text{ is right-unique}
 \end{aligned} \tag{4.10.1}$$

4.11 Relational diagram

A relational diagram shows three sets: the sets A and B as well as the relation R . The elements of A and B are represented by different symbols, for instance empty and filled circles. The elements of R are represented by line segments. For $R \subseteq A \times B$ the elements $a \in A$ and $b \in B$ for which aRb is true are joined by line segments. The following relational diagrams illustrate the uniqueness of R .

**Figure 4.1:** Uniqueness of R

4.12 Types of relations

Every relation is a subset of a direct product of sets. Relations often have additional properties. Relations with common properties belong to a type of relations. Some types of relations are defined in the following.

4.12.1 Identity relation

The set of all ordered pairs (a, a) in the product $A \times A$ is called the identity relation I_A in the set A .

$$I_A := \{(a, a) \mid a \in A\} \quad (4.12.1)$$

4.12.2 Inverse relation

The set R^{-1} is called the inverse (dual) relation of the relation R if the order of the elements in the ordered pairs (a, b) of R is exchanged in R^{-1} .

$$R^{-1} := \{(b, a) \mid (a, b) \in R\} \quad (4.12.2)$$

4.12.3 Composition

Let a relation R on the sets A and B and a relation S on the sets B and C be given. The set of ordered pairs $(a, c) \in A \times C$ for which there is a common element in B is called the composition of R and S . The order of R and S is relevant, as b is the second element of R and the first element of S . The composition is designated by $R \circ S$.

$$R \circ S := \left\{ (a, c) \in A \times C \mid \bigvee_{b \in B} (aRb \wedge bSc) \right\} \quad (4.12.3)$$

4.12.4 Equivalence relation

A relation $E \subseteq M \times M$ is called an equivalence relation in the set M if it is reflexive, symmetric and transitive. The elements x and y of the set M are said to be equivalent if the set E contains the pair (x, y) ; this relationship is designated by $x \sim y$ or xEy .

$$\begin{array}{lll} E \text{ is reflexive} & x \sim x & \\ E \text{ is symmetric} & x \sim y \Rightarrow y \sim x & \\ E \text{ is transitive} & x \sim y \wedge y \sim z \Rightarrow x \sim z & \end{array} \quad (4.12.4)$$

4.12.5 Equivalence class

A subset of a set M is called an equivalence class in M if the elements of the subset are pairwise equivalent. An equivalence class is designated by choosing an arbitrary element a of the class and enclosing it in square brackets $[a]$. The selected element a is called a representative of its class.

$$[a] := \{x \in M \mid (a, x) \in E\} \quad (4.12.5)$$

4.12.6 Partitioning by equivalence

The equivalence classes in a set M for a given equivalence relation E form a partition of M :

1. Every element x of the set M is contained in at least one equivalence class, since (x, x) is an element of the reflexive relation E .
2. None of the equivalence classes $[x]$ is empty, since $(x, x) \in E$ and hence at least x itself is an element of $[x]$.
3. Every element z of the set M is contained in exactly one equivalence class. In fact, if z is an element of the classes $[x]$ and $[y]$, then since E is symmetric and transitive $z \sim x$ and $z \sim y$ imply that $x \sim z$ and $x \sim y$; hence $[x] = [y]$.

4.12.7 Quotient set

The set of equivalence classes of a set M for an equivalence relation E is called a quotient set and is designated by M/E (M modulo E). A subset $R \subseteq M$ is called a system of representatives of the quotient set M/E if it contains exactly one representative from each class of M/E .

$$M/E := \{[x] \mid x \in M\} \quad (4.12.6)$$

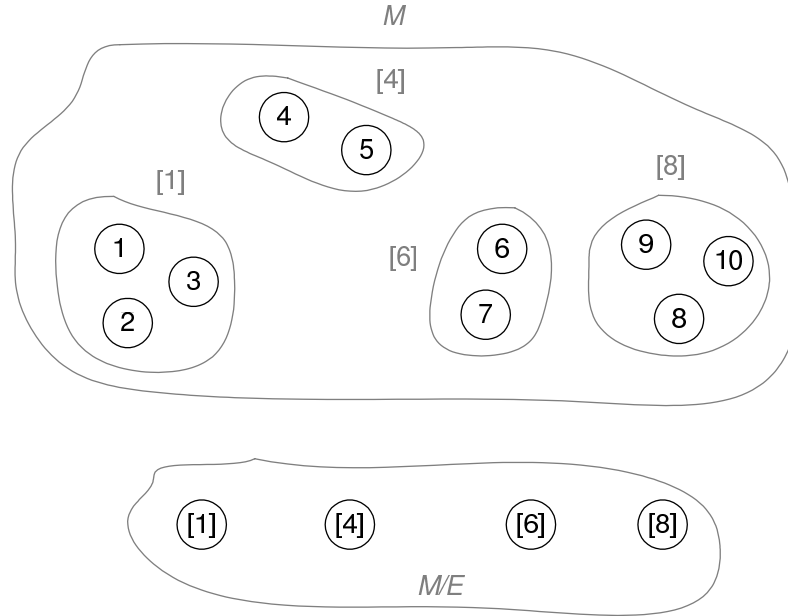


Figure 4.2: Quotient set

4.13 Mappings

It may be convenient to assign to each element of a set A exactly one element of a set Z . The same element of Z may be assigned to different elements of A . Relations of this type are called mappings. Each vertex can be mapped in this way to a vertex in its strongly connected component.

4.13.1 Mapping notation

A relation $\Phi \subseteq A \times Z$ is called a mapping if it is left-total and right-unique. The terminology used is:

$$\begin{array}{ll} \Phi : A \rightarrow Z & \Phi \text{ is a mapping from } A \text{ of } Z \\ A & \text{domain of } \Phi \\ Z & \text{target of } \Phi \end{array} \quad (4.13.1)$$

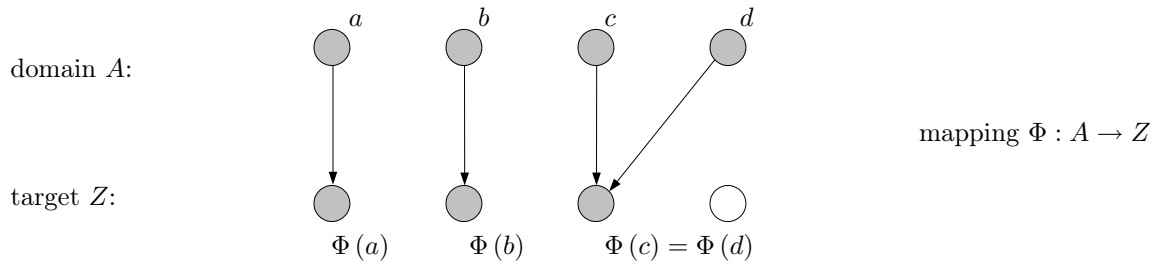
4.13.2 Image of an element

If the mapping Φ assigns the element $z \in Z$ to the element $a \in A$, then z is called the image of a under the mapping Φ . The element a is called a inverse image (pre-image) of z . The following notation is used:

$$\Phi : a \rightarrow z \quad \text{or} \quad \Phi(a) = z \quad (4.13.2)$$

4.13.3 Arrow diagram

Mappings are depicted using arrow diagrams. Every element of the domain is the starting point of an arrow. The arrow points to the image in the target.

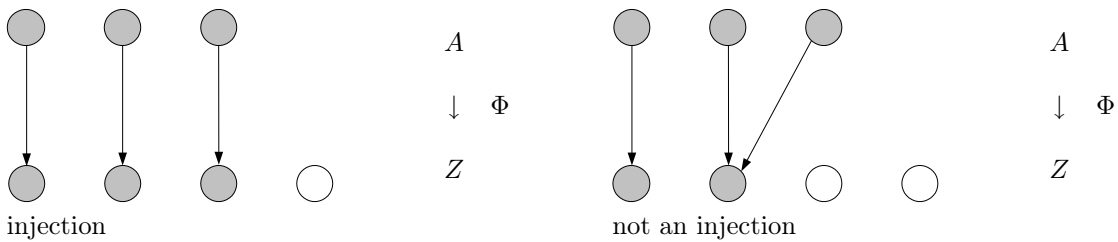


4.13.4 Types of mappings

All mappings are left-total and right-unique relations. Mappings often have additional properties. Mappings with common additional properties are classified as follows.

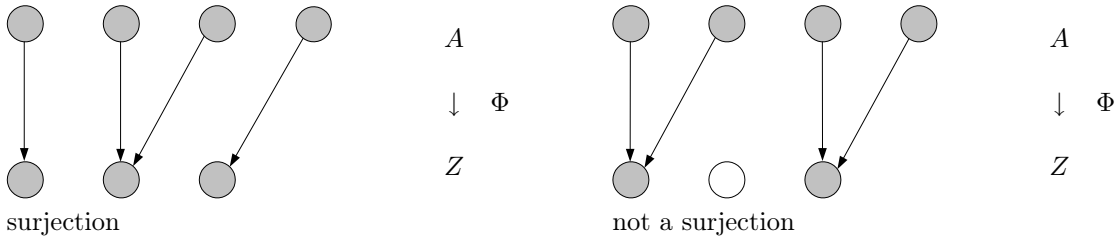
4.13.4.1 Injective mapping

A mapping $\Phi : A \rightarrow Z$ is said to be injective (an injection) if two different elements $a \neq b$ of the set A always possess two different images $\Phi(a) \neq \Phi(b)$. An injection is a left-total, bi-unique relation. From $\Phi(a) = \Phi(b)$ it follows that $a = b$.



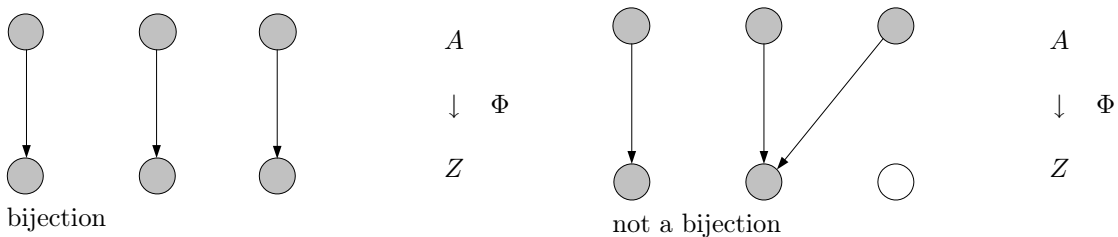
4.13.4.2 Surjective mapping

A mapping $\Phi : A \rightarrow Z$ is said to be surjective (a surjection) if each element of the target Z is the image of at least one element of A . A surjection is a bitotal, right-unique relation. An element $z \in Z$ may be the image of more than one element in A .



4.13.4.3 Bijective mapping

A mapping $\Phi : A \rightarrow Z$ is said to be bijective (a bijection) if every element of Z is the image of exactly one element of A . A bijection is a bitotal, bi-unique relation. The number of elements in A and Z is the same.



4.13.4.4 Canonical mapping

The surjection from a set M to its quotient set M/E for a given equivalence relation E is called a canonical mapping of M . The image of the element $a \in M$ is the equivalence class $[a]$.

$$k : M \rightarrow M/E \quad \text{with} \quad k(a) = [a] \quad (4.13.3)$$

Example

The graph example in section 4.17 will be used to show the canonical mapping between the graph M and its quotient set M/E .

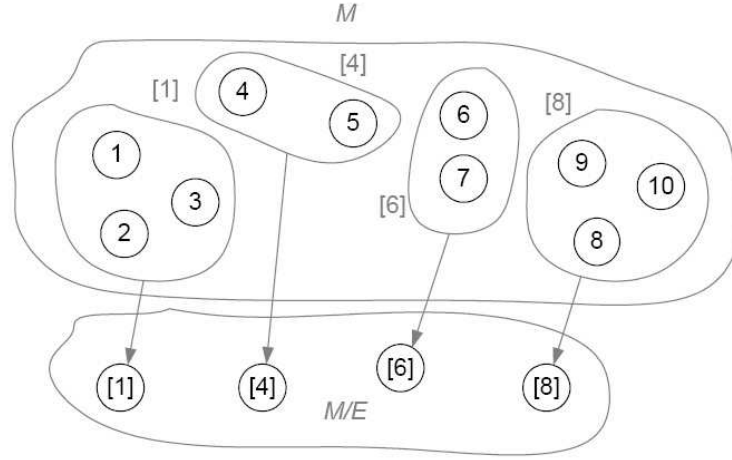


Figure 4.3: Canonical mapping

4.14 Order relations and ordered sets

To order a set an order relation is defined. The definition of an order relation does not require that the elements of a set be comparable. The set can be partially ordered. For the special case of a total ordering of a set all elements of the set need to be comparable. Order relations (e.g. \leq in \mathcal{N}), the set of natural numbers) as well as strict order relations (e.g. $<$ in \mathcal{N}) may be used to order a set either partially or totally.

Order relation

For a given set M a relation in M is defined as an order relation if it is reflexive, antisymmetric and transitive. Order relations are represented by symbols such as \leq or \subseteq . An order relation \leq in M is a subset of the Cartesian product $M \times M$ with the properties listed below for elements $a, b, c \in M$.

$$\begin{array}{lll}
 \leq & \text{is reflexive} & : a \in M \Rightarrow a \leq a \\
 \leq & \text{is antisymmetric} & : a \leq b \wedge b \leq a \Rightarrow a = b \\
 \leq & \text{is transitive} & : a \leq b \wedge b \leq c \Rightarrow a \leq c
 \end{array} \tag{4.14.1}$$

4.15 Countability and cardinality

4.15.1 Cardinal numbers and finite and infinite sets

The number of elements in a finite set is the cardinal number of the set.

The concept which corresponds to the number of elements of a finite set for general sets (which can be finite or infinite) is also termed the cardinal number or cardinality of the set.

For a given collection of sets $M = \{A, B, \dots\}$ the quotient set with respect to the equivalence relation \sim is formed. An element of the quotient set M/\sim is called the cardinal number or cardinality of the given collection of sets. The canonical mapping $\text{card} : M \rightarrow M/\sim$ assigns a cardinality $\text{card}(A) = [A] = |A|$ to each set $A \in M$.

4.15.2 Operations on cardinal numbers

Operations on cardinal numbers of a given set of sets can be defined as follows.

- The sum of the cardinal numbers of disjoint sets A and B is the cardinal number of the union of A and B :
 $A \cap B = \phi \Rightarrow \text{card}(A) + \text{card}(B) = \text{card}(A \cup B)$
- The product of the cardinal numbers of sets A and B is the cardinal number of the Cartesian product of A and B :
 $\text{card}(A) \cdot \text{card}(B) = \text{card}(A \times B)$
- The cardinal numbers $\text{card}(A)$ to the power $\text{card}(B)$ is the cardinal number of the set of all mappings from B to A :
 $\text{card}(A)^{\text{card}(B)} = \text{card}(A^B) = \text{card}\{f \mid f : B \rightarrow A\}$

4.15.3 Countable sets and properties of countable sets

A set is defined as countable if an injection $f : M \rightarrow \mathcal{N}$ exists. \mathcal{N} is the set of natural numbers $\mathcal{N} = \{1, 2, 3, 4, 5 \dots\}$

$$M \text{ is countable} :\Leftrightarrow \bigvee_f (f : M \rightarrow \mathcal{N} \text{ is injective}) \quad (4.15.1)$$

Countable sets have the following properties:

- The Cartesian product $\mathcal{N} \times \mathcal{N}$ of the set of natural numbers $\mathcal{N} = \{1, 2, 3, 4, 5 \dots\}$ is countable.
- For every injection $f : A \rightarrow B$ with $A \neq \phi$ there is a surjection $g : B \rightarrow A$ with $g \circ f = 1_A$ the identity mapping.
- If a set A is countable and a mapping $f : A \rightarrow B$ is surjective then it follows that the set B is countable.
- Every subset of a countable set is countable.
- If the sets A and B are countable then the Cartesian product of A and B , $A \times B$ is countable.
- The union of countable sets is countable.

4.15.4 Comparison of and ordering of cardinal numbers

To compare the cardinal numbers of sets the order relation *less than or equal to* is used.

Define $S = \{A, B, C \dots\}$ be a collection of sets. The set S is partitioned into classes of equipotent sets using the equivalence relation \sim . The quotient set S / \sim is the set of cardinal numbers for S . The cardinal number of A is said to be *less than or equal to* the cardinal number of B if A is equipotent with a subset of $C \subseteq B$ in S .

$$\text{card}(A) \leq \text{card}(B) :\Leftrightarrow \bigvee_{C \in S} (C \subseteq B \wedge A \sim C) \quad (4.15.2)$$

It follows that the cardinal numbers of the sets in S are partially ordered by the less than or equal to relation (\leq) since \leq possesses the properties of an order relation, i.e.:

- The relation \leq is reflexive since for $A \sim A$:
 $\text{card}(A) \in S / \sim \Rightarrow \text{card}(A) \leq \text{card}(A)$
- The relation \leq is antisymmetric:
 $\text{card}(A) \leq \text{card}(B) \wedge \text{card}(B) \leq \text{card}(A) \Rightarrow \text{card}(A) = \text{card}(B)$
- The relation \leq is transitive:
 $\text{card}(A) \leq \text{card}(B) \wedge \text{card}(B) \leq \text{card}(C) \Rightarrow \text{card}(A) \leq \text{card}(C)$

4.15.5 Cardinality of the set of real numbers

It can be proved that no bijective mapping exists between the set of real numbers R and the set of natural numbers N . It follows that $\text{card}(N) \leq \text{card}(R)$

4.15.6 Cardinality of Cartesian products of the set of real numbers

Define the open interval $I = (0, 1)$. It can be shown that $\text{card}(I) = \text{card}(R)$ and thus $\text{card}(I^2) = \text{card}(R^2)$ and thus:

$$\text{card}(R^2) = \text{card}(R) \Leftrightarrow \text{card}(I^2) = \text{card}(I). \quad (4.15.3)$$

By use of induction it can then be shown that $\text{card}(R^n) = \text{card}(R)$.

4.16 Closure of a homogeneous binary relation

An extension of a homogeneous binary relation $R \subseteq M \times M$ is called a closure and is designated by $\langle R \rangle$ if the following conditions are satisfied:

$$\begin{aligned} \text{inclusion} & : R \subseteq \langle R \rangle \\ \text{isotonicity} & : R \subseteq S \Rightarrow \langle R \rangle \subseteq \langle S \rangle \\ \text{idempotency} & : \langle \langle R \rangle \rangle = \langle R \rangle \end{aligned} \quad (4.16.1)$$

The extension is performed such that the closure has special properties which the relation itself does not necessarily possess. Reflexive, symmetric and transitive closures are defined in the following sections. Closures may also have several of these properties.

4.16.1 Reflexive closure

The reflexive closure $\langle R \rangle_r$ of a relation $R \subseteq M \times M$ is formed by adding the elements $(x, x) \in M \times M$ to R . The closure $\langle R \rangle_r$ satisfies the condition for reflexive relations.

$$\begin{aligned} \langle R \rangle_r & := \{(x, y) \mid (x, y) \in R \vee x = y \in M\} \\ \langle R \rangle_r & = R \sqcup I \\ I & \subseteq \langle R \rangle_r \Rightarrow \langle R \rangle_r \text{ is reflexive} \end{aligned} \quad (4.16.2)$$

4.16.2 Symmetric closure

The symmetric closure $\langle R \rangle_s$ of a relation $R \subseteq M \times M$ is the union of R with its transpose R^T . If $\langle R \rangle_s$ contains the element (x, y) , then (y, x) is also an element of $\langle R \rangle_s$. The closure $\langle R \rangle_s$ satisfies the condition for symmetric relations.

$$\begin{aligned} \langle R \rangle_s & := \{(x, y) \mid (x, y) \in R \vee (y, x) \in R\} \\ \langle R \rangle_s & = R \sqcup R^T \\ \langle R \rangle_s & = \langle R \rangle_s^T \Rightarrow \langle R \rangle_s \text{ is symmetric} \end{aligned} \quad (4.16.3)$$

4.16.3 Powers of a relation

In the algebra of relations, connections are represented by products of the relation R with itself. For example, if R contains the elements (a, b) and (b, c) , then by definition the product $R \circ R$ contains the element (a, c) . The element (a, c) is a connection of length 2 in R . Each of the elements of $R \circ R$ is a connection of length 2 in R . The power $R^m = R \circ \dots \circ R$ (m -fold) contains all connections of length m between two elements of M . To determine all connections of length $m \leq q$ in M by R , the union of the relations $R \sqcup R^2 \sqcup \dots \sqcup R^q$ is formed.

4.16.4 Stability index

The least exponent s for which the union $R \sqcup R^2 \sqcup \dots \sqcup R^s$ is not changed by adding terms R^m with $m > s$ is called the stability index of the relation R . The union $R \sqcup R^2 \sqcup \dots \sqcup R^s$ contains all connections by R in M .

The stability index s of a relation R may be interpreted as follows: If there are several connections between two elements of M , then there is a shortest connection of length q which is contained in R^q .

Among all the shortest connections between pairs of elements, there is a shortest connection of maximal length s , which is contained in the power R^s . Hence the union $R \sqcup R^2 \sqcup \dots \sqcup R^s$ contains all connections in M by R . For a set M with n elements, the stability index s of the relation $R \subseteq M \times M$ is less than n , since the maximal length of all shortest connections in M by R cannot be greater than $n - 1$.

4.16.5 Transitive closure

The transitive closure $\langle R \rangle_t$ of a relation $R \subseteq M \times M$ contains all elements $(x, y) \in M \times M$ which are connected in M by R . The closure $\langle R \rangle_t$ satisfies the condition for transitive closures.

$$\begin{aligned} \langle R \rangle_t &:= \{(x, y) \in M \times M \mid x \text{ and } y \text{ are connected in } M \text{ by } R\} \\ \langle R \rangle_t &:= R \sqcup \dots \sqcup R^s \\ \langle R \rangle_t \circ \langle R \rangle_t &\subseteq \langle R \rangle_t \Rightarrow \langle R \rangle_t \text{ is transitive} \\ s \text{ stability index of } R \text{ with } \langle R \rangle_t \sqcup R^{s+1} &= \langle R \rangle_t \end{aligned} \quad (4.16.4)$$

4.16.6 Reflexive transitive closure

The reflexive transitive closure $\langle R \rangle_{rt}$ of a relation $R \subseteq M \times M$ may alternatively be regarded as the transitive closure $\langle \langle R \rangle_r \rangle_t$ of the reflexive closure $\langle R \rangle_r$ or as the reflexive closure $\langle \langle R \rangle_t \rangle_r$ of the transitive closure $\langle R \rangle_t$. The two viewpoints lead to identical relations. The closure $\langle R \rangle_{rt} = \langle R \rangle$ satisfies the condition for transitive relations in the special form of an equation.

$$\begin{aligned} \langle R \rangle_{rt} &:= \langle \langle R \rangle_r \rangle_t & \langle R \rangle_{tr} &:= \langle \langle R \rangle_t \rangle_r \\ \langle R \rangle_{rt} &= \langle R \rangle_{tr} \\ \langle R \rangle_{rt} \circ \langle R \rangle_{rt} &= \langle R \rangle_{rt} \Rightarrow \langle R \rangle_{rt} \text{ is transitive} \end{aligned} \quad (4.16.5)$$

4.16.7 Reflexive symmetric transitive closure

The reflexive symmetric transitive closure $\langle R \rangle_{rst}$ of a relation $R \subseteq M \times M$ is the transitive closure of the symmetric closure of the transitive closure of R . It coincides with the reflexive symmetric transitive closure $\langle R \rangle_{srt}$. The closure $\langle R \rangle_{rst}$ is of special importance, as it is an equivalence relation and therefore yields a classification of the set M .

$$\begin{aligned} \langle R \rangle_{rst} &:= \langle \langle \langle R \rangle_r \rangle_s \rangle_t = \langle \langle R \rangle_s \rangle_{rt} = \langle R \sqcup R^T \rangle_{rt} \\ \langle R \rangle_{rst} &= \langle R \sqcup I \sqcup R^T \rangle_t \end{aligned} \quad (4.16.6)$$

4.17 Algebra of homogeneous binary relations

Directed graphs will be considered in chapter 5.

Since the edge set of a directed graph is a homogeneous binary relation on the vertex set, the properties of homogeneous binary relations and their rules of calculation may be directly transferred to directed graphs. To support the graph theory development which follows the algebra of homogeneous binary relations will now be explained in more detail.

Since every relation is a set, the rules of the algebra of sets also hold for homogeneous binary relations. Additional properties and rules result from the duality and composition of relations.

4.17.1 Graphical representation

A homogeneous binary relation R in a set M can be visually represented in a graph diagram. The graph diagram consists of a point set which represents the set M of elements with their designations. If an element x is related to an element y , an arrow is drawn from the point x to the point y . The homogeneous relation R corresponds to the resulting set of arrows. The graph diagram shows the elements of the set M and the relationships in a network-like structure. It is the representation used in graph theory. The points used to represent the elements are called vertices, the arrows are called directed edges.

The directed edges of the graph can be labelled r_{ij} with i and j indicating the start and end vertex for the graph edge r_{ij} .

Example

$$\begin{aligned}
M &= \{a, b, c, d, e\} \\
R &= \{(a, b), (a, d), (b, a), (c, a), (c, d), (d, c), (d, e), (e, e)\}
\end{aligned}$$

A graphical representation of the given homogeneous binary relation R in the set M is shown in figure 4.4.

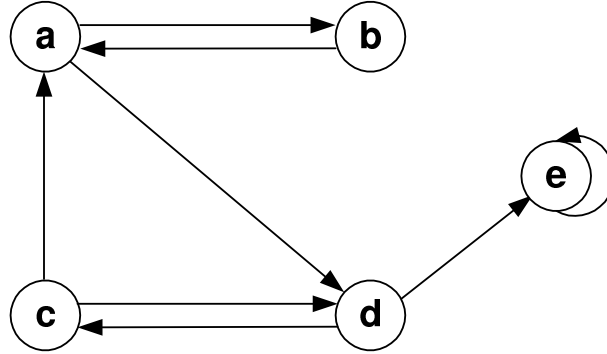


Figure 4.4: Homogeneous binary relations graph example

4.17.2 Special relations

The null relation (empty relation) ϕ , the identity relation I and the all relation (universal relation) E are special homogeneous binary relations in the set M .

$$\begin{aligned}
\text{null relation} \quad \phi &= \{\} \\
\text{identity relation} \quad I &= \{(a, a) \mid a \in M\} \\
\text{all relation} \quad E &= M \times M
\end{aligned} \tag{4.17.1}$$

4.17.3 Equality and inclusion

The equality $R = S$ and inclusion $R \subseteq S$ operations on homogeneous relations R and S are equivalent. If $R \subseteq S$ is true, then R is contained in S .

The equality and inclusion operations transfer to directed graphs \mathbf{R} and \mathbf{S} as shown. r_{ij} and s_{ij} represent the edges of the directed graphs \mathbf{R} and \mathbf{S} respectively.

$$\begin{aligned}
\text{equality} \quad R = S &: \Leftrightarrow \bigwedge_{a,b} ((a, b) \in R \Leftrightarrow (a, b) \in S) \\
\text{inclusion} \quad R \subseteq S &: \Leftrightarrow \bigwedge_{a,b} ((a, b) \in R \Rightarrow (a, b) \in S) \\
\text{equality} \quad \mathbf{R} = \mathbf{S} &: \Leftrightarrow \bigwedge_{i,j} (r_{ij} \Leftrightarrow s_{ij}) \\
\text{inclusion} \quad \mathbf{R} \subseteq \mathbf{S} &: \Leftrightarrow \bigwedge_{i,j} (r_{ij} \Rightarrow s_{ij})
\end{aligned} \tag{4.17.2}$$

4.17.4 Binary operations

The intersection $R \sqcap S$, the union $R \sqcup S$ and the product $R \circ S$ are binary operations on the homogeneous relations R and S . The intersection and the union are defined as in set theory. The product corresponds to the composition of two relations; the operation of forming products is called multiplication. In the algebra of relations it is convenient to define the composition $R \circ S$ of the relations in the order ‘first R , then S ’. This definition allows direct transfer to boolean matrix algebra.

$$\begin{array}{lll}
\text{intersection} & R \sqcap S & := \{(x, y) \mid (x, y) \in R \wedge (x, y) \in S\} \\
\text{union} & R \sqcup S & := \{(x, y) \mid (x, y) \in R \vee (x, y) \in S\} \\
\text{product} & R \circ S & := \{(x, y) \mid \bigvee_z ((x, z) \in R \wedge (z, y) \in S)\}
\end{array} \tag{4.17.3}$$

The intersection, union and product operations transfer to directed graphs **R** and **S** as shown below. r_{ij} and s_{ij} represent the edges of the directed graphs **R** and **S** respectively.

$$\begin{array}{lll}
\text{intersection} & \mathbf{R} \sqcap \mathbf{S} & := [r_{ij} \wedge s_{ij}] \\
\text{union} & \mathbf{R} \sqcup \mathbf{S} & := [r_{ij} \vee s_{ij}] \\
\text{product} & \mathbf{R} \circ \mathbf{S} & := \left[\bigvee_k r_{ik} \wedge s_{kj} \right]
\end{array} \tag{4.17.4}$$

4.18 MATLAB implementation of Relation Operations

Refer to Appendix C for the MATLAB implementation of basic relation operations with examples.

Chapter 5

Graph Theory

5.1 Introduction

The basic concepts and terminology of graph theory which may be applied to business systems modelling are reviewed in this chapter. The development follows from works by Pahl and Damrath [92], Chartrand and Oellerman [23], Balakrishnan [11], Gross and Yellen [51], Gross and Yellen [52], Lipschutz [74] and Cronje [26]. Selected paragraphs taken from these references are used as such.

5.2 Graphs and Directed graphs

A variety of models of real-world situations can be represented by means of a diagram consisting of a set of points and a set of lines or curves linking some or all of these points.

Graph theory is the mathematical abstraction dealing such structures of points and lines.

The lines linking points may be directed, which gives rise to the concept of a directed graph or digraph.

5.3 Graphs

A graph G consists of a set V of vertices and a collection E (not necessarily a set) of unordered pairs of vertices called edges. A graph is symbolically represented as $G = (V; E)$.

Typically V and E are finite unless defined otherwise. The order of a graph is the number of vertices and the size is the number of edges. If u and v are two vertices of a graph and if the edge $e = \{u, v\}$ is defined, then e is said to join u and v or is the edge between u and v . u and v is said to be incident on e and e is incident on both u and v . An edge which is incident on the same vertex is called a loop. Note that $\{u, v\}$ is an unordered pair of u and v .

A graph with undirected edges joining any given two vertices and not having any loops is called a simple graph.

5.4 Graph isomorphism

Identical graphs

Two graphs $G = (V; E)$ and $G' = (V'; E')$ are identical if $V = V'$ and $E = E'$. This rigid approach is typically too restrictive and the structural equivalence between two non-equivalent graphs lead to the concept of isomorphic graphs.

Isomorphic graphs

Two graphs $G = (V; E)$ and $G' = (V'; E')$ are isomorphic if a bijective map or isomorphism f exists from V to V' such that an edge exists between $f(u)$ and $f(v)$ in G' if and only if there is an edge between u and v in G . Equivalent graphs are isomorphic.

5.5 Subgraphs

The graph $H = (W; F)$ is a subgraph of graph $G = (V; E)$ if W is a subset of V ($W \subseteq V$) and F is a subset of E ($F \subseteq E$).

5.6 Directed graphs

A directed graph (digraph) is a structured set. It consists of the vertex set V and a homogeneous binary vertex relation R which corresponds to a set of directed edges. The vertex set V is equipped with structure by the vertex relation R . The structural properties of a directed graph are entirely determined by the properties of the relation R .

The relationships between the vertices are called edges of the graph and are identified by an ordered vertex pair. Therefore, the edge set is a homogeneous binary relation on the vertex set. The properties of homogeneous binary relations and their rules of calculation (see section 4.7) may be directly transferred to directed graphs.

5.6.1 Definition of a directed graph

A domain $G := (V; R)$ is called a directed graph if V is the vertex set and $R \subseteq V \times V$ is the edge set of the graph. An edge from the vertex $x \in V$ to the vertex $y \in V$ is designated by the ordered pair $(x, y) \in R$. The edge, also known as an arc, (x, y) is said to be directed from x to y . The vertex x is called the start vertex of the edge. The vertex y is called the end vertex of the edge.

$$\begin{array}{ll} G & := (V; R) \quad R \subseteq V \times V \\ V & \text{set of vertices} \\ R & \text{set of ordered vertex pairs (edge set)} \end{array} \quad (5.6.1)$$

The graph G is called a null graph if the vertex set is empty. It is called an empty graph if the edge set is empty. It is called a complete graph if the edge set R is the all relation $E = V \times V$.

5.6.2 Properties

The properties of a directed graph $(V; R)$ are determined by the properties of the homogeneous binary relation R . The properties of homogeneous relations described in Table 4.1 are therefore transferred to directed graphs in Table 5.1. Antireflexive, symmetric, antisymmetric and asymmetric graphs are important in applications:

Table 5.1: Properties of directed graphs

G is antireflexive	$:\Leftrightarrow$	$I \subseteq \bar{R}$
G is symmetric	$:\Leftrightarrow$	$R = R^T$
G is antisymmetric	$:\Leftrightarrow$	$R \cap R^T \subseteq I$
G is asymmetric	$:\Leftrightarrow$	$R \cap R^T = \emptyset$

For an antireflexive graph, the edge set does not contain vertex pairs of the form (x, x) , and the graph diagram is free of loops (see section 5.9.1.6). Between two different vertices in the graph diagram, a symmetric graph contains either no edge or a pair of edges with opposite directions, which are combined into an undirected edge. An antisymmetric graph contains either no edges or only one directed edge between two vertices in the graph diagram. Symmetric and antisymmetric graphs may contain loops. An asymmetric graph is antisymmetric and antireflexive, and hence free of loops. The graphs we will be considering are asymmetric.

5.7 Degrees, indegrees and outdegrees

If a graph vertex v has p loops incident to it as well as q other edges incident to it the degree of v is $2p + q$. In a graph with no loops, the degree of a vertex is the number of edges adjacent to that vertex. In a graph without loops an isolated vertex has degree 0 and an end-vertex has degree 1. The sum of the degrees in a graph is twice the number of edges of the graph. The maximum degree of a graph G is denoted by $\Delta(G)$ and the minimum degree $\delta(G)$. A vertex in a graph is an odd vertex if its degree is odd or an even vertex when its degree is even. Every graph has an even number of odd vertices.

In a directed graph the sum of the outdegrees of all the vertices is equal to the number of arcs which is also equal to the sum of all the indegrees of the vertices.

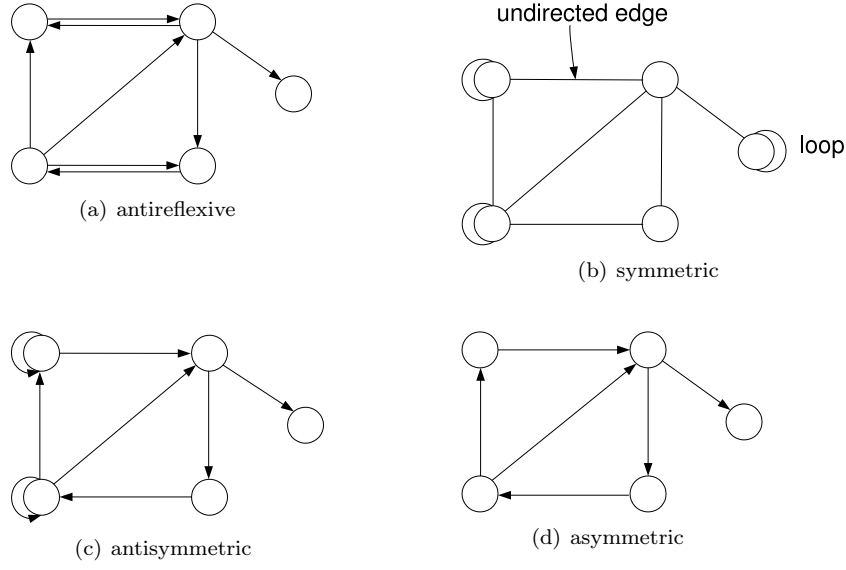


Figure 5.1: Directed graph properties

5.7.1 Equality and inclusion

Let two directed graphs G_1 and G_2 be given. Using the algebra of relations, equality and inclusion are defined as follows for these graphs:

$$\begin{array}{lll}
 \text{equality} & G_1 = G_2 & :\Leftrightarrow V_1 = V_2 \wedge R_1 = R_2 \\
 \text{partial graph} & G_1 \sqsubseteq G_2 & :\Leftrightarrow V_1 = V_2 \wedge R_1 \sqsubseteq R_2 \\
 \text{subgraph} & G_1 \subseteq G_2 & :\Leftrightarrow V_1 \subseteq V_2 \wedge R_1 \subseteq R_2 \quad \square \quad (V_1 \times V_1)
 \end{array} \tag{5.7.1}$$

A partial graph or spanning subgraph G_1 is generated from a graph G_2 by removing edges from G_2 . A subgraph G_1 is generated from a graph G_2 by first removing vertices together with the incident edges and then removing further edges from G_2 .

5.7.2 Adjacency matrix graph representation

Graphs can be represented using different data structures, one of which is the adjacency matrix.

Let V be a set with n elements. The elements of V are indexed by a mapping $\Phi : N \rightarrow V$ with $\Phi(i) = x_i$ and $1 \leq i \leq n$, so that $V = \{x_1, \dots, x_n\}$. A homogeneous binary relation $R \subseteq V \times V$ is a subset of $V \times V$. The elements of $V \times V$ which belong to the relation are specified by a boolean matrix \mathbf{R} of dimension $n \times n$. Every element $(x_i, x_j) \in V \times V$ is bijectively associated with an element $r_{ij} \in \mathbf{R}$. If the relation R contains the element (x_i, x_j) , then r_{ij} has the value true(1); otherwise r_{ij} has the value false (0).

A boolean matrix \mathbf{R} of a homogeneous relation R is an n^2 -tuple of the truth values $W = \{0, 1\}$, and hence an element of the n^2 -fold Cartesian product $W^{n \cdot n}$. The elements of a matrix \mathbf{R} are usually arranged in a row and column scheme by regarding the indices i, j of the element r_{ij} as row and column indices, respectively. In formulations of general properties and rules, a matrix \mathbf{R} is represented by a general element r_{ij} in square brackets.

$$\mathbf{R} = [r_{ij}] = \begin{bmatrix} r_{11} & \cdots & r_{1j} & \cdots & r_{1n} \\ \vdots & & \vdots & & \vdots \\ r_{i1} & \cdots & r_{ij} & \cdots & r_{in} \\ \vdots & & \vdots & & \vdots \\ r_{n1} & \cdots & r_{nj} & \cdots & r_{nn} \end{bmatrix} \quad \begin{array}{l} W = \{0, 1\} \\ \mathbf{R} \in W^{n \cdot n} \end{array} \tag{5.7.2}$$

5.8 Graph representation and manipulation

For computational and visualisation processing purposes graphs can be represented using adjacency matrices or incidence matrices.

5.8.1 Adjacency matrices

The adjacency matrix of a graph $G = (V; E)$ where $V = \{1, 2, 3, 4, \dots, n\}$ is the $n \times n$ symmetric matrix $\mathbf{A} = [a_{ij}]$ where the non-diagonal entries of A , a_{ij} are the number of edges joining vertex i and vertex j and the diagonal entries a_{ii} are twice the number of loops at vertex i . The adjacency matrix of a simple graph is a boolean matrix with zero entries along the diagonal.

For the adjacency matrix of a graph the sum of the entries in a row or column corresponding to a vertex is equal to the degree of the vertex. The sum of all the entries in the matrix is twice the number of edges in the graph.

The adjacency matrix of a digraph $G = (V; E)$ where $V = \{1, 2, 3, 4, \dots, n\}$ is the $n \times n$ boolean matrix $A = [a_{ij}]$ where the non-diagonal entry of A , $a_{ij} = 1$ if and only there is an arc from vertex i to vertex j . The diagonal entry a_{ii} is zero for all i .

For the adjacency matrix of a digraph the sum of the entries in a row corresponding to a vertex is equal to the outdegree of the vertex and the sum of the entries in a column corresponding to a vertex is equal to the indegree of the vertex. The sum of all the entries in the matrix is equal to the number of arcs in the digraph.

Adjacency matrices – examples

The adjacency matrix of the graph in figure 5.2 is given by:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}.$$

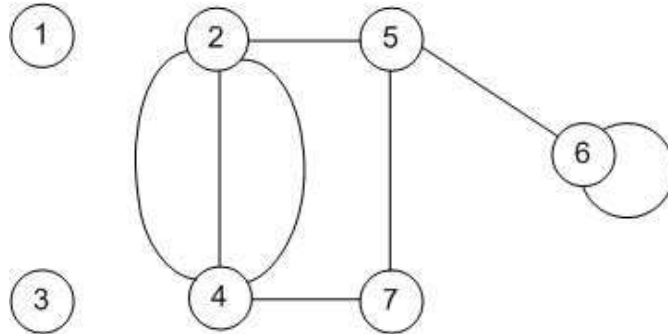


Figure 5.2: Graph example

The adjacency matrix of the digraph in figure 5.3 is given by:

$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

5.8.2 Incidence matrices

If $G = (V, E)$ is a simple graph with $V = \{1, 2, 3, 4, \dots, n\}$ and $E = \{e_1, e_2, e_3, e_4, \dots, e_m\}$ the incidence matrix $\mathbf{B} = [b_{ij}]$ ($n \times m$) of G is defined as follows. Row i of B corresponds to vertex i for each i . Column k corresponds to edge e_k for each k . If edge e_k is incident on vertex i and j the entries $b_{ik} = b_{jk} = 1$ and all other entries in column k are zero.

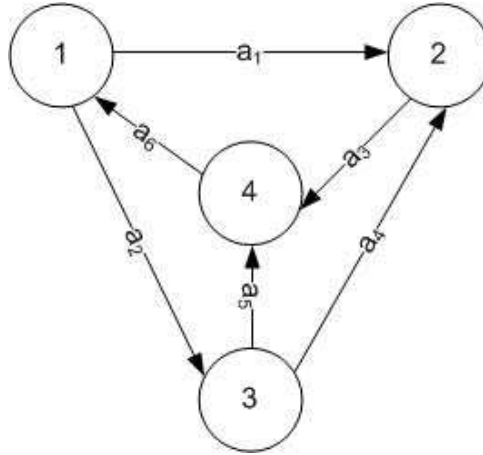


Figure 5.3: Directed graph example

If G is a digraph and e_k is the arc from vertex i to vertex j the entries of incidence matrix $B = [b_{ij}]$ are defined as $b_{ik} = -1$ (from vertex) and $b_{jk} = 1$ (to vertex) in column k . All other entries in column k are zero.

For the incidence matrix of a simple graph the sum of the entries in a row corresponding to a vertex is equal to the degree of the vertex. The sum of all the entries in the matrix is twice the number of edges in the graph.

For the incidence matrix of a digraph the sum of the entries in a row corresponding to a vertex is equal to the outdegree minus the indegree of the vertex. The sum of all the entries in the incidence matrix is equal to zero.

Incidence matrices – examples

The incidence matrix of the digraph in figure 5.3 is given by:

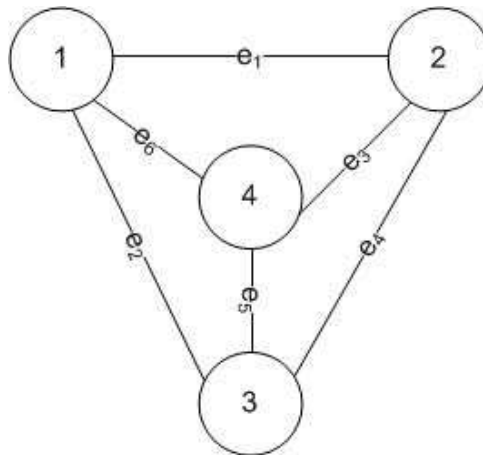
$$\begin{bmatrix} -1 & -1 & 0 & 0 & 0 & 1 \\ 1 & 0 & -1 & 1 & 0 & 0 \\ 0 & 1 & 0 & -1 & -1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -1 \end{bmatrix}.$$


Figure 5.4: Simple graph example

The incidence matrix of the simple graph in figure 5.4 is given by:

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}.$$

5.9 Structure of graphs

The structure of a graph is uniquely determined by the relations of the domain. The structure of the directed graph $G = (V; R)$ is determined by the edge relation R . A review of paths and cycles in graphs and graph connectedness follows.

5.9.1 Paths and cycles in directed graphs

Paths and cycles in directed graphs are examples of subgraphs. The definition of paths and cycles in a directed graph forms the basis of the structural analysis of graphs. The existence of paths and cycles between two vertices leads to the formation of the transitive closure R^+ of the relation R . The properties of the transitive closure allow a classification into acyclic, anticyclic and cyclic graphs.

5.9.1.1 Predecessor and successor

A vertex x is called a predecessor of a vertex y if there is an edge from x to y in the graph, so that the ordered vertex pair (x, y) is contained in the relation R . If x is a predecessor of y , then y is called a successor of x .

$$\begin{aligned} x \text{ predecessor of } y &\Leftrightarrow (x, y) \in R &\Leftrightarrow \\ y \text{ successor of } x &\Leftrightarrow (x, y) \in R^T \end{aligned} \quad (5.9.1)$$

A vertex x in a vertex set V may be regarded as a unary point relation in V . In the following, this unary point relation is also designated by x . The predecessorship and the successorship of vertices $x, y \in V$ are formulated as an inclusion using such unary relations:

$$\begin{aligned} x \text{ predecessor of } y &\Leftrightarrow xy^T \subseteq R &\Leftrightarrow \\ y \text{ successor of } x &\Leftrightarrow yx^T \subseteq R^T \end{aligned}$$

The set of all predecessors of a vertex $x \in V$ is designated by $t_p(x)$ and the set of all successors of x by $t_s(x)$. The sets $t_p(x)$ and $t_s(x)$ are unary relations in V and are determined as follows using the edge relation R :

$$\begin{aligned} \text{predecessors of } x : t_p(x) &= Rx \\ \text{successors of } x : t_s(x) &= R^T x \end{aligned}$$

5.9.1.2 Indegree and outdegree for vertices in paths

The number of predecessors of a vertex x is called the indegree of x and is designated by $g_p(x)$. The indegree of $g_p(x)$ corresponds to the number of elements in the set $t_p(x)$, and hence to the number of directed edges which end at the vertex x . The number of successors of a vertex x is called the outdegree of x and is designated by $g_s(x)$. The outdegree $g_s(x)$ corresponds to the number of elements in the set $t_s(x)$, and hence to the number of directed edges which emanated from the vertex x .

$$\begin{aligned} \text{indegree } g_p(x) &= |t_p(x)| = |Rx| \\ \text{outdegree } g_s(x) &= |t_s(x)| = |R^T x| \end{aligned} \quad (5.9.2)$$

The sum of the indegrees of all vertices $x \in V$ is equal to the number of directed edges of the directed graph, and hence coincides with the number of elements of the relation R . The same is true for the outdegrees.

$$\sum_{x \in V} g_p(x) = \sum_{x \in V} g_s(x) = |R|$$

5.9.1.3 Edge sequence

A chain of edges is called an edge sequence if the end vertex of each edge except for the last edge is the start vertex of the following edge.

$$\begin{aligned} &\langle (x_0, x_1), (x_1, x_2), \dots, (x_{n-1}, x_n) \rangle \\ &\quad \bigwedge_{j=1}^n ((x_{j-1}, x_j) \in R) \end{aligned} \quad (5.9.3)$$

The start vertex x_0 of the first edge and the end vertex x_n of the last edge are called the start vertex and the end vertex of the edge sequence, respectively. The vertices x_1 to x_{n-1} are called intermediate vertices of the edge sequence. The number n of edges is called the length of the edge sequence. An edge may occur more than once in an edge sequence.

5.9.1.4 Ancestors and descendants

A vertex x is called an n^{th} ancestor of a vertex y if there is an edge sequence of length n from x to y in the graph. If x is an n^{th} ancestor of y , then y is called an n^{th} descendant of x . A 1^{st} ancestor or 1^{st} descendant of x is a predecessor or successor of x , respectively. The n^{th} ancestors and descendants of x are determined recursively from the relationships for predecessors and successors according to the following rule:

n^{th} ancestors of x :

$$\begin{aligned} t_p^{(k)}(x) &= R t_p^{(k-1)}(x) & \text{for } k = 1, \dots, n & \text{ with } t_p^{(0)}(x) = x \\ t_p^{(n)}(x) &= R^n x & \text{for } n > 0 \end{aligned} \quad (5.9.4)$$

n^{th} descendants of x :

$$\begin{aligned} t_s^{(k)}(x) &= R^T t_s^{(k-1)}(x) & \text{for } k = 1, \dots, n & \text{ with } t_s^{(0)}(x) = x \\ t_s^{(n)}(x) &= (R^n)^T x & \text{for } n > 0 \end{aligned} \quad (5.9.5)$$

The set of all ancestors of a vertex x is designated by $t_p^+(x)$; it is determined as the union of the sets of n^{th} ancestors of x . The set $t_s^+(x)$ of all descendants of x is determined analogously. The transitive closure R^+ of a relation R with stability index s , may be used to determine these sets:

ancestors of x :

$$t_p^+(x) = t_p^{(1)}(x) \sqcup \dots \sqcup t_p^{(s)}(x) = Rx \sqcup \dots \sqcup R^s x = R^+ x$$

descendants of x :

$$t_s^+(x) = t_s^{(1)}(x) \sqcup \dots \sqcup t_s^{(s)}(x) = Rx \sqcup \dots \sqcup R^s x = R^+ x$$

5.9.1.5 Path

A path from a start vertex x via intermediate vertices to an end vertex y is an edge sequence. In a directed graph, a path may be uniquely represented as a vertex sequence $\langle x, \dots, y \rangle$. A path $\langle x \rangle$ with the same start and end vertex x contains no edges and is called an empty path. The length of an empty path is 0. There is an empty path for every vertex of a directed graph. The existence of non-empty paths in a directed graph is established as follows:

$$\begin{aligned} \text{there is a path of length } n \text{ from } x \text{ to } y & \Leftrightarrow xy^T \subseteq R^n \\ \text{there is a non-empty path from } x \text{ to } y & \Leftrightarrow xy^T \subseteq R^+ \end{aligned} \quad (5.9.6)$$

5.9.1.6 Cycle

A non-empty path whose start vertex and end vertex coincide is called a cycle. A loop at a vertex is a cycle of length 1. A cycle which contains no loops is called a proper cycle. If there is a non-empty path from x to y and a non-empty path from y to x , then the concatenation of the two paths yields a cycle through x and y . The existence of cycles in a directed graph is established as follows:

$$\begin{aligned} \text{there is a cycle of length } n > 0 \text{ through } x & \Leftrightarrow xx^T \subseteq R^n \\ \text{there is a cycle through } x & \Leftrightarrow xx^T \subseteq R^+ \\ \text{there is a cycle through } x \text{ and } y & \Leftrightarrow xy^T \subseteq R^+ \cap R^{+T} \end{aligned} \quad (5.9.7)$$

5.9.1.7 Acyclic graph

A directed graph $G = (V; R)$ is said to be acyclic if it does not contain any cycles. The transitive closure R^+ of an acyclic graph is asymmetric. If there is a non-empty path from x to y , then there is no non-empty path from y to x , since otherwise the concatenation of the two paths would yield a cycle.

$$R^+ \cap R^{+T} = 0 \quad (5.9.8)$$

The acyclicity of a graph leads to special structural properties of the graph. Directed acyclic graphs possess an order structure. The vertex set is an ordered set. The directed edges describe the order relation in the vertex set. Due to the order structure, the vertices can be sorted topologically.

5.9.1.8 Anticyclic graph

A directed graph $G = (V; R)$ is said to be anticyclic if it does not contain any proper cycles. In contrast to acyclic graphs, an anticyclic graph may contain loops at the vertices. The transitive closure R^+ of an anticyclic graph is antisymmetric.

$$R^+ \cap R^{+T} \subseteq I \quad (5.9.9)$$

5.9.1.9 Cyclic graph

A directed graph $G = (V; R)$ is said to be cyclic if every non-empty path in G belongs to a cycle. The transitive closure R^+ of a cyclic graph is symmetric. If there is a non-empty path from x to y , then there is also a non-empty path from y to x , so that the concatenation of the two paths yields a cycle.

$$R^+ = R^{+T} \quad (5.9.10)$$

5.9.1.10 Properties

The following relationships hold between the properties of a relation R and of its transitive closure R^+ . If the transitive closure R^+ is asymmetric or antisymmetric, then the relation R is asymmetric or antisymmetric, respectively. If the relation R is symmetric, then the transitive closure R^+ is asymmetric or antisymmetric, then the relation R is asymmetric or antisymmetric, respectively. If the relation R is symmetric, then the transitive closure R^+ is symmetric. These relationships lead to the following implications:

$$\begin{array}{ll} \text{acyclic graph} & \Rightarrow \text{asymmetric graph} \\ \text{anticyclic graph} & \Rightarrow \text{antisymmetric graph} \\ \text{cyclic graph} & \Leftarrow \text{symmetric graph} \end{array}$$

5.9.1.11 Simple path

A non-empty path is said to be simple if it does not contain any edge more than once. The vertices and the edges of a simple path form a subgraph of the directed graph. If the start vertex and end edge of a simple path are different, the following relationships hold between the indegrees and the outdegrees of the vertices of the corresponding subgraph.

For a subgraph for a simple path $\langle x, \dots, z, \dots, y \rangle$ with $x \neq y$:

$$\begin{array}{ll} \text{start vertex} & g_s(x) = g_p(x) + 1 \\ \text{intermediated vertex} & g_s(z) \\ \text{end vertex} & g_p(y) - 1 \end{array} \quad (5.9.11)$$

5.9.1.12 Simple cycle

A simple path whose start vertex and end vertex coincide is called a simple cycle. In the subgraph for a simple cycle, the indegree and the outdegree of each vertex are equal.

For a subgraph for a simple cycle with vertex z :

$$\text{vertex } g_s(z) = g_p(z) \quad (5.9.12)$$

5.9.1.13 Elementary path

A non-empty path is said to be elementary if it does not contain any vertex more than once. The vertices and the edges of an elementary path form a subgraph. If the start vertex and the end vertex

of an elementary path are different, then the vertices of the corresponding subgraph have the following indegrees and outdegrees.

For subgraph for an elementary path $\langle x, \dots, z, \dots, y \rangle$ with $x \neq y$:

$$\begin{array}{lll} \text{start vertex} & g_s(x) = 1 & g_p(x) = 0 \\ \text{intermediated vertex} & g_s(z) = 1 & g_p(z) = 1 \\ \text{end vertex} & g_p(y) = 0 & g_p(y) = 1 \end{array} \quad (5.9.13)$$

5.9.1.14 Elementary cycle

An elementary path whose start vertex and end vertex coincide is called an elementary cycle. In the subgraph of an elementary cycle, the indegree and the outdegree of every vertex are equal to 1. Note that the identical start and end vertex is counted once, not twice.

For a subgraph for an elementary cycle with vertex z :

$$\text{vertex } g_s(z) = g_p(z) = 1 \quad (5.9.14)$$

5.9.2 Connectedness of directed graphs

In a directed graph, a vertex may or may not be reachable from another vertex along the directed edges. The concept of reachability forms the basis for a definition of the connectedness of vertices. Different kinds of connectedness may be defined, such as strong and weak connectedness. Directed graphs which are not strongly or weakly connected may be decomposed uniquely into strongly or weakly connected subgraphs. These subgraphs are called strongly or weakly connected components, respectively. The decomposition of a graph into its strongly connected components (see section 5.9.2.8) leads to an acyclic reduced graph.

5.9.2.1 Reachability

In a directed graph $G = (V; R)$, a vertex $y \in V$ is said to be reachable from a vertex $x \in V$ if there is an empty or non-empty path from x to y . Vertex y is reachable from vertex x if and only if the product xy^T of the associated point relations x and y is contained in the reflexive transitive closure R^* .

$$y \text{ is reachable from } x : \Leftrightarrow xy^T \subseteq R^* \quad R^* = I \sqcup R^+ \quad (5.9.15)$$

5.9.2.2 Strong connectedness

Two vertices x and y of a directed graph are said to be strongly connected if x is reachable from y and y is reachable from x . A directed graph is said to be strongly connected if all vertices are pairwise strongly connected.

$$\begin{array}{ll} x \text{ and } y \text{ are strongly connected} & : \Leftrightarrow xy^T \subseteq R^* \sqcup R^{*T} \\ \text{the graph is strongly connected} & : \Leftrightarrow R^* \cap R^{*T} = E \Leftrightarrow R^* = E \end{array} \quad (5.9.16)$$

5.9.2.3 Unilateral Connectedness

Two vertices x and y of a directed graph are said to be unilaterally connected if x is reachable from y or y is reachable from x . A directed graph is said to be unilaterally connected if all vertices are pairwise unilaterally connected.

$$\begin{array}{ll} x \text{ and } y \text{ are unilaterally connected} & : \Leftrightarrow xy^T \subseteq R^* \sqcup R^{*T} \\ \text{the graph is unilaterally connected} & : \Leftrightarrow R^* \cap R^{*T} = E \end{array} \quad (5.9.17)$$

5.9.2.4 Weak connectedness

Two vertices x and y of a directed graph $(V; R)$ are said to be weakly connected if they are strongly connected in the symmetric graph $G = (V; R \sqcup R^T)$. A directed graph is said to be weakly connected if all vertices are pairwise weakly connected. Since the transitive closure of a symmetric relation is symmetric, this definition may be expressed as follows:

$$\begin{array}{ll} x \text{ and } y \text{ are weakly connected} & : \Leftrightarrow xy^T \subseteq (R \sqcup R^T)^* \\ \text{the graph is weakly connected} & : \Leftrightarrow (R \sqcup R^T)^* = E \end{array} \quad (5.9.18)$$

5.9.2.5 Connectedness relations

The relation R of a directed graph $G = (V; R)$ generally contains strong, unilateral and weak connections. A relation which contains only connections of the same type is called a connectedness relation. The connectedness relations for a directed graph G are derived from the relation R and its reflexive transitive closure R^* :

$$\begin{array}{ll} \text{strong connectedness relation} & S = R^* \sqcap R^{*T} \\ \text{unilateral connectedness relation} & P = R^* \sqcup R^{*T} \\ \text{weak connectedness relation} & C = (R \sqcup R^T)^* \end{array} \quad (5.9.19)$$

A strongly connected vertex pair is also unilaterally connected; a unilaterally connected vertex pair is also weakly connected. Hence a strongly connected graph is also unilaterally connected, and a unilaterally connected graph is also weakly connected. For a symmetric graph, the three different kinds of connectedness coincide.

$$\begin{array}{lll} \text{inclusion} & : & R^* \sqcap R^{*T} \subseteq R^* \sqcup R^{*T} \subseteq (R \sqcup R^T)^* \\ \text{connectedness} & : & \text{strong} \Rightarrow \text{unilateral} \Rightarrow \text{weak} \end{array}$$

Two different vertices which are strongly connected lie on a cycle. A strongly connected graph is therefore cyclic. The converse is not true in the general case.

$$\text{strongly connected graph} \Rightarrow \text{cyclic graph}$$

5.9.2.6 Properties of the connectedness relations

The strong connectedness relation S is reflexive, symmetric and transitive. Reflexivity and symmetry follow directly from the definition. Transitivity follows from the following consideration. If (x, y) and (y, z) are strongly connected vertex pairs, then z is reachable from x via y and x is reachable from z via y . Hence (x, z) is also a strongly connected vertex pair.

The unilateral connectedness relation P is reflexive and symmetric, but generally not transitive. This follows from the following consideration. If (x, y) and (y, z) are unilaterally connected vertex pairs, then it is possible that x is only reachable from y and z is only reachable from y . In this case, neither is x reachable from z , nor is z reachable from x . Thus (x, z) is not a unilaterally connected vertex pair.

The weak connectedness relation C is by definition the strong connectedness relation of an associated symmetric graph. This is reflexive, symmetric and transitive.

A reflexive, symmetric and transitive relation is an equivalence relation. Hence the strong and weak connectedness relations are equivalence relations. The unilateral connectedness relation is generally not an equivalence relation.

5.9.2.7 Decomposition into connected components

A graph may be decomposed into subgraphs which have simple structural characteristics and yield insight into the essential structural properties of the graph.

The strong connectedness relations $S = (R \sqcup R^T)^*$ of a directed graph $G = (V; R)$ are equivalence relations. Let Z stand for either of these equivalence relations. The graph $(V; R)$ is connected if the equivalence relation Z is the all relation E . If the graph $(V; R)$ is disconnected, then it may be uniquely decomposed into connected subgraphs. The subgraphs are called the connected components of the graph. The decomposition is carried out in the following steps, independent of the kind of connectedness being considered:

1. Connectedness class: The vertex set V of the graph is partitioned into connected classes, using the relation Z . A connected class $[x]$ with the vertex x as a representative contains all vertices of V which are connected with x . The class $[x]$ is a unary relation and is determined as follows:

$$[x] = Zx \quad (5.9.20)$$

2. Mapping: The set K of all connected classes is the quotient set V/Z . Each vertex $x \in V$ is mapped to exactly one connected class, yielding a canonical mapping Φ :

$$\Phi : V \rightarrow K \quad \text{with} \quad K = V/Z \quad (5.9.21)$$

3. Reduced graph: The mapping Φ from the vertex set V of the directed graph $G = (V; R)$ to the set K of connected classes induces the reduced graph $G_K = (K; R_K)$.

$$G_K = (K; R_K) \quad \text{with} \quad R_K = \Phi^T R \Phi \quad (5.9.22)$$

4. Connected component: A connected component is a connected subgraph $G_k := (V_k, R_k)$ of a directed graph $G = (V; R)$. The vertex set V_k contains all vertices of a connected class K of the graph $(V; R)$. The edge set $R_k = R \cap (V_k \times V_k)$ contains the edges from R whose vertices belong to V_k . The union of all connected components G_k is generally a partial graph of G , since the union of all vertex sets V_k is the vertex set V and the union of all edge sets R_k is only a subset of the edge set R .

$$\bigsqcup_{k \in K} G_k \subseteq G \quad (5.9.23)$$

5.9.2.8 Decomposition into strongly connected components

The vertex set V of a directed graph $G = (V; R)$ may be decomposed into strongly connected classes using its strong connectedness relation $Z = S = R^* \cap R^{*T}$. Two different classes cannot be strongly connected in the reduced graph $G_K = (K; R_K)$, since strongly connected vertices belong to the same class. Each connected component $G_k = (V_k; R_k)$ has a symmetric transitive closure R_k^+ and is therefore a cyclic graph. The reduced graph $G_K = (K; R_K)$ has an antisymmetric transitive closure R_K^+ and is therefore an anticyclic graph.

5.9.2.9 Decomposition into weakly connected components

The vertex set V of a directed graph $G = (V; R)$ may be decomposed into weakly connected classes using its weak connectedness relation $Z = C = (R \sqcup R^T)^*$. Two different classes cannot be weakly connected in the reduced graph $G_K = (K; R_K)$, since weakly connected vertices belong to the same class and the two vertices of an edge are at least weakly connected. Hence every directed graph is the union of its weakly connected components.

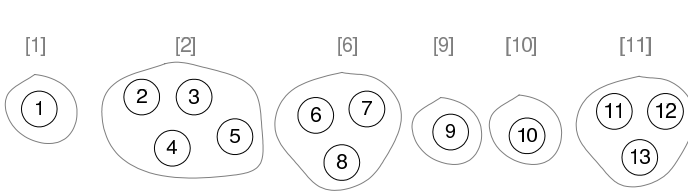
$$G = \bigsqcup_{k \in K} G_k \quad (5.9.24)$$

5.9.2.10 Strongly connected components example

Graph

The directed graph in Figure 5.5 will be used to demonstrate the decomposition of a directed graph into its strongly connected components.

Strongly connectedness classes



$$\begin{aligned} [x] &= Sx \\ [1] &= \{1\} \\ [2] &= \{2, 3, 4, 5\} \\ [6] &= \{6, 7, 8\} \\ [9] &= \{9\} \\ [10] &= \{10\} \\ [11] &= \{11, 12, 13\} \end{aligned}$$

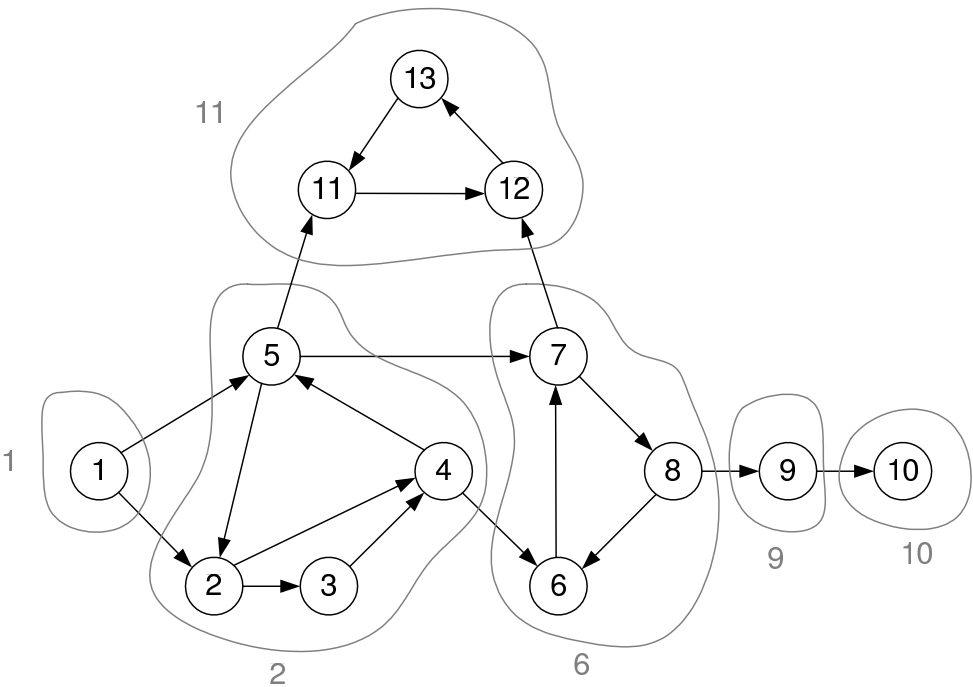
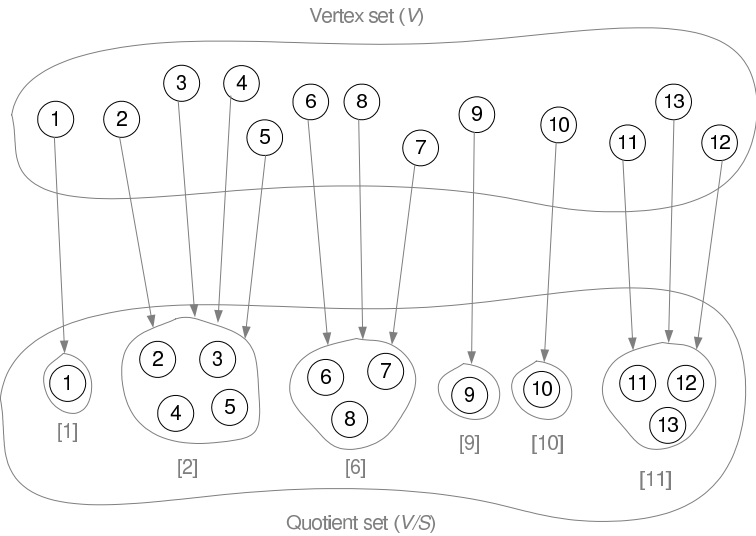


Figure 5.5: Strongly connected components graph

Mapping

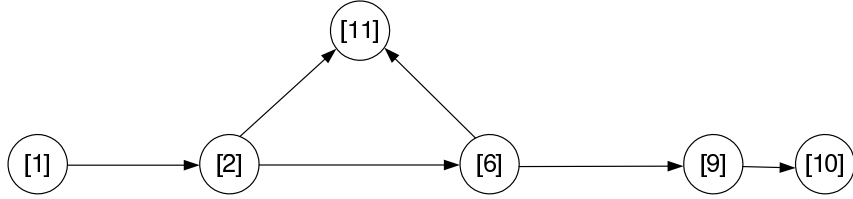


$\Phi : V \rightarrow V/S$	
1	$\rightarrow \{1\}$
2	$\rightarrow \{2, 3, 4, 5\}$
3	$\rightarrow \{2, 3, 4, 5\}$
4	$\rightarrow \{2, 3, 4, 5\}$
5	$\rightarrow \{2, 3, 4, 5\}$
6	$\rightarrow \{6, 7, 8\}$
7	$\rightarrow \{6, 7, 8\}$
8	$\rightarrow \{6, 7, 8\}$
9	$\rightarrow \{9\}$
10	$\rightarrow \{10\}$
11	$\rightarrow \{11, 12, 13\}$
12	$\rightarrow \{11, 12, 13\}$
13	$\rightarrow \{11, 12, 13\}$

Reduced graph

$$G_K = (K; R_K)$$

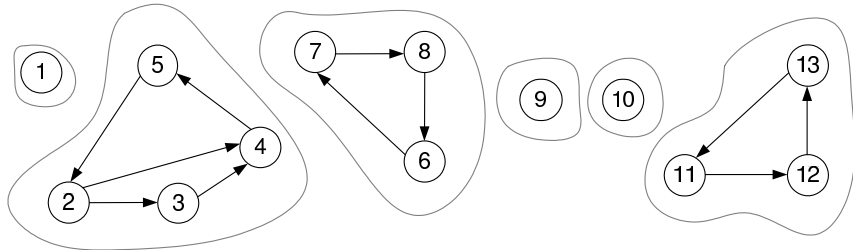
Vertex set (K) $\{[1], [2], [6], [9], [10], [11]\}$
 Edge set (R_K) $\{([1], [2]), ([2], [6]), ([2], [11]), ([6], [9]), ([9], [10])\}$

**Figure 5.6:** *Reduced graph***Strongly connected components**

$$\bigsqcup_{k \in K} G_k \subseteq G$$

Table 5.2: Strongly connected components

Strongly connected component	Vertex set	Edge set
[1]	{1}	-
[2]	{2, 3, 4, 5}	{(2, 3), (2, 4), (3, 4), (4, 5), (5, 2)}
[6]	{6, 7, 8}	{(6, 7), (7, 8), (8, 6)}
[9]	{9}	-
[10]	{10}	-
[11]	{11, 12, 13}	{(11, 12), (12, 13), (13, 11)}

**Figure 5.7:** *Strongly connected components***5.9.3 Acyclic graphs****5.9.3.1 Directed acyclic graph**

A directed acyclic graph $G = (V; R)$ is asymmetric and does not contain cycles. Every path from a vertex x to a vertex y is elementary. The closure R^+ is asymmetric and transitive. Hence it is a strict order relation. The theoretical foundations of strict order relations may therefore be applied to directed acyclic graphs.

5.9.3.2 Rank

Every vertex x of a directed acyclic graph $G = (V; R)$ is assigned a rank $r(x)$, which is a natural number with the following properties:

1. A vertex x has the rank $r(x) = 0$ if it does not have any ancestors.
2. A vertex x has the rank $r(x) = k > 0$ if it has a k^{th} ancestor and no $(k+1)^{th}$ ancestors.

It is only possible to assign ranks if the directed graph G is acyclic. If there is a cycle through the vertex x , then for every k^{th} ancestor of x in the cycle there is a predecessor in the cycle, and hence also a $(k+1)^{th}$ ancestor of x . The directed graph must therefore be free of cycles.

If the rank $r(x)$ of a vertex x is k , then by definition the vertex x has a k^{th} ancestor but no $(k+1)^{th}$ ancestor. Thus there must be a path of length k but no path of length $k+1$ from a vertex without predecessor in G to x . Hence the rank $r(x)$ is the length k of a longest path from a vertex without predecessor in G to x .

5.9.3.3 Topological Sorting

The determination of the ranks of the vertices of a directed graph $G = (V; R)$ is called topological sorting. The vertex set $V = V_0$ is topologically sorted by iteratively reducing it to the empty vertex set \emptyset . In step k , the vertex set V_k is determined whose vertices $x \in V_k$ have a k^{th} ancestor in G and are therefore of rank $r(x) \geq k$. The vertex set V_k contains all predecessors of the vertices in the vertex set V_{k-1} . This iterative reduction is formulated as follows using unary relations:

$$\begin{array}{llll} \text{initial values} & : & v_0 & = & e & \text{all relation} \\ \text{reduction} & : & v_k & = & R^T v_{k-1} & k = 1, \dots, n \\ \text{termination} & : & v_n & = & \emptyset & \text{null relation} \end{array} \quad (5.9.25)$$

A vertex x of the vertex set V_k is of degree $r(x) = k$ if it does not belong to the vertex set V_{k+1} . The set W_k of all vertices of rank k is therefore the difference $V_k - V_{k+1}$, which is calculated as the intersection of V_k and the complement of V_{k+1} . It is called the k^{th} vertex class and is determined as a unary relation as follows:

$$w_k = v_k \cap \bar{v}_{k+1} \quad k = 0, \dots, n-1 \quad (5.9.26)$$

5.9.3.4 Order structure

Topologically sorting a directed acyclic graph $G = (V; R)$ yields a partition of the vertex set into disjoint vertex classes W_k with $k = 0, \dots, n-1$. The partition has the following ordinal properties:

- The vertex class W_0 contains all vertices of the lowest rank 0. These vertices have no ancestors in G , and hence no predecessors. They are therefore minimal. Since there are no other vertices without predecessors, W_0 contains all minimal vertices.
- The vertex class W_{n-1} contains all vertices of the highest rank $n-1$. These vertices have no descendants in G , and hence no successors. They are therefore maximal. Since there may generally also be other vertices without successor, W_{n-1} generally does not contain all maximal vertices.
- Every vertex x in the vertex class W_k with $k > 0$ has at least one predecessor y in the vertex class W_{k-1} . If $x \in W_k$ did not have a predecessor $y \in W_{k-1}$, then x would not have any k^{th} ancestors, and would therefore not belong to W_k .
- A vertex has neither a predecessor nor a successor in its own vertex class. If y were a predecessor of x and hence x a successor of y , then the rank of y would have to be less than the rank of x and x, y could not belong to the same vertex class.

5.9.3.5 Basic edges and chords

A directed acyclic graph $G = (V; R)$ has basic edges and chords. An edge (x, y) in the directed graph G is called a basic edge if y is reachable from x only via this edge. If the basic edge is removed, then y is no longer reachable from x .

An edge (x, y) in the directed graph G is called a chord if the vertex y is also reachable from the vertex x via other edges. The chord (x, y) is the shortest path from x to y .

Since a directed acyclic graph does not contain cycles, an edge from x to y is a chord if and only if there is a path of length $n > 1$ from x to y .

$$\begin{array}{ll}
 \text{path from } x \text{ to } y \text{ with } n > 1 & \Leftrightarrow xy^T \subseteq \bigsqcup_{n>1} R^n = R \bigsqcup_{n>0} R^n = RR^+ \\
 \text{chord } (x, y) & \Leftrightarrow xy^T \subseteq R \cap RR^+ \\
 \text{basic arc } (x, y) & \Leftrightarrow xy^T \subseteq R \cap \overline{RR^+}
 \end{array} \tag{5.9.27}$$

5.9.3.6 Basic path

A directed acyclic graph $G = (V; R)$ does not contain cycles. If there are one or more paths from x to y , then there is at least one path of maximal length. A path of maximal length is called a basic path. A basic path contains only basic edges.

5.9.3.7 Basic graph

The graph $B = (V; Q)$ is a basic graph of a directed acyclic graph $G = (V; R)$ if Q contains only the basic edges in R . The basic graph B is constructed by removing all chords from R . The basic graph B is unique. The transitive closures R^+ and Q^+ coincide.

$$B = (V; Q) \quad \text{with} \quad Q = R \cap \overline{RR^+} \tag{5.9.28}$$

5.9.3.8 Order diagram

In the topological sorting of a directed acyclic graph $G = (V; R)$, the rank $r(x)$ of a vertex $x \in V$ is equal to the length of a longest path from a vertex without predecessor to x . This path is a basic path consisting only of basic edges. Hence removing chords from R does not change the rank $r(x)$ of a vertex x , so that topologically sorting the graph $G = (V; R)$ and its basic graph $B = (V; Q)$ leads to the same result. The representation of the order structure of the basic graph with its vertex classes is an order diagram.

5.9.4 Simple acyclic graphs and trees

A simple acyclic graph $G = (V; \Gamma)$ is a simple graph which does not contain any cycles. All the undirected edges of the graph G are bridges. If a bridge is removed to form a simply connected graph, the graph is divided into two simply connected components and is not connected any more.

5.9.4.1 Trees

A simple acyclic graph which is simply connected is called a tree.

A tree with n vertices has exactly $n - 1$ undirected edges.

number of vertices	n
number of edges	k
for a tree	$n - k = 1$

For a tree the path between two different vertices x and y is unique. If different paths between x and y existed, cycles would exist in the graph, which by definition can then not be a tree.

5.9.4.2 Forests

A simple acyclic graph with several simply connected components is called a forest. Every simply connected component of the forest is a tree. Each component of a forest is a tree and any tree is a connected forest.

A forest with n vertices and k undirected edges contains exactly $n - k$ trees.

number of vertices	n
number of edges	k
number of components for a forest	c
thus for a forest	$n - k = c$

5.10 Rooted graphs and rooted trees

A vertex of a graph from which all remaining vertices are reachable is called a root of the graph. All hierarchical structures can be regarded as rooted trees. Searching for all vertices of a graph which are reachable from a given vertex leads to a search tree which corresponds to a rooted tree and forms a skeleton of the graph.

5.10.1 Root

A vertex w is called a root (root vertex) of a directed graph $G = (V; R)$ if all vertices of the graph are reachable from the vertex w . If a directed graph is not weakly connected, then it has no root. If it is strongly connected, then every vertex of the graph is a root.

$$w \text{ is a root} \quad :\Leftrightarrow \quad we^T \subseteq R^* \quad (5.10.1)$$

R^* is the closure of R .

5.10.2 Rooted graphs

A directed graph $G = (V; R)$ is called a rooted graph if it contains at least one root. In a rooted graph, there is a special form of connectedness between pairs of vertices, called quasi-strong connectedness. Two vertices x and y are quasi-strongly connected if there is a vertex z from which the vertices x and y are both reachable. In this case, there is a path from x to z in the dual graph G^T and a path from z to y in the graph G , so that $(x, z) \in R^{*T}$ and $(z, y) \in R^*$, and hence $(x, y) \in R^{*T}R^*$. In a rooted graph, all vertices are pairwise quasi-strongly connected via a root, so that $R^{*T}R = E$ holds.

$$\begin{aligned} x \text{ and } y \text{ are quasi-strongly connected} & \quad :\Leftrightarrow \quad xy^T \subseteq R^{*T}R^* \\ G = (V; R) \text{ is a rooted graph} & \quad :\Leftrightarrow \quad R^{*T}R^* = E \end{aligned} \quad (5.10.2)$$

5.10.3 Acyclic rooted graphs

A directed graph $G = (V; R)$ is acyclic if $R^+ \cap R^{+T} = \phi$ holds. It is a rooted graph if $R^{*T}R^* = E$ holds. An acyclic rooted graph has exactly one root. The existence of several roots would contradict the absence of cycles.

$$G = (V; R) \text{ is an acyclic rooted graph} \quad \Leftrightarrow \quad R^+ \cap R^{+T} = \phi \wedge R^{*T}R^* = E \quad (5.10.3)$$

5.10.4 Rooted trees

An acyclic rooted graph $G = (V; R)$ is called a rooted tree if R is left-unique, so that $RR^T \subseteq I$ holds.

$$G = (V; R) \text{ is a rooted tree} \quad :\Leftrightarrow \quad RR^T \subseteq I \wedge R^+ \cap R^{+T} = \phi \wedge R^{*T}R^* = E \quad (5.10.4)$$

A rooted tree with the root w has the following properties:

- The root w has no predecessor.
- Every vertex $x \neq w$ has exactly one predecessor.
- Every vertex $x \neq w$ is reachable along exactly one path from w to x .
- A rooted tree with n vertices has exactly $n - 1$ edges.

5.10.5 Forest of rooted trees

A directed graph is called a forest of rooted trees if every weakly connected component is a rooted tree.

5.10.6 Search tree

Let a vertex a in a directed graph G be given. A rooted tree with root a which contains all descendants of a in G is called a search tree at the vertex a . A search tree is constructed by an iterative search, starting from the vertex a . Breadth-first search and depth-first search are distinguished.

5.11 Depth-first search

5.11.1 Depth-first search for trees and forests

The vertices and some of the edges of a directed graph form a depth-first search tree during the depth-first search. The depth-first search tree is a representation of the order in which the vertices had been visited. Only edges pointing to previously unvisited vertices are part of a depth-first search tree. Therefore, each depth-first search tree is a directed acyclic subgraph of the directed graph. Depth-first search trees for directed graphs are rooted trees. The number of depth-first search trees formed during a depth-first search depends on the order in which the vertices are visited, as well as the structure of the graph. If more than one depth-first search tree is formed, we have a depth-first search forest. Different depth-first searches, with different depth-first search forests can be done on the same graph, depending on the order in which vertices are visited.

5.11.2 Pre-order and post-order numbering

During the depth-first search, pre- and post-order numbers are assigned to each vertex. The pre-order numbers indicate the order in which the vertices are first visited, while the post-order numbers indicate the order in which vertices are finished with in the depth-first search.

5.11.3 Classification of edges

The edges of a directed graph can be classified into four groups during a depth-first search. The classification of an edge is a property of both the structure of the graph and the dynamics of the search. Since there is more than one depth-first search forest for each graph, different classifications may be given to an edge of a graph for different depth-first searches. The pre- and post-order numbers are used to classify the edges.

Tree edges correspond to a recursive call in the depth-first search, i.e. the start vertex has been visited, but the end vertex has not been visited before. Tree edges are the edges of the depth-first search trees. The other types of edges are not part of the depth-first search tree. (Start vertex pre-order number = -1 .)

Back edges indicate that the directed graph contains at least one cycle. The number of back edges does not necessarily correspond to the number of cycles in the directed graph. The start vertex of a back edge has been visited previously. The end vertex has also been visited previously, and is also an ancestor of the start vertex in the depth-first search tree. The removal of all the back edges results in a directed acyclic graph. (End vertex pre-order number = -1 .)

Down edges The start vertex of a down edge points to a previously visited end vertex, which is a descendant of the start vertex in the depth-first search tree. Down edges are also known as chords (see section 5.9.3.5) in the directed graph. (Start vertex pre-order number $>$ end vertex pre-order number.)

Cross edges The start vertex of a cross edge, points to a previously visited end vertex, which is neither an ancestor nor a descendant of the start vertex in the depth-first search tree. Cross edges connect vertices in different depth-first search trees (If it is not a tree, back or down edge.)

Example

The edge classifications can be seen visually in Figure 5.8. The back edge $(3, 1)$ is an indication of a cycle in the graph, in this case cycle $(1, 2), (2, 3), (3, 1)$. A down edge is an indication of a chord in the graph, in this case, if the chord $(2, 5)$ is cut, vertex 5 will still be reachable from vertex 2, via vertex 4. A cross edge points from a vertex in one depth-first search tree, vertex 7, to a vertex in another depth-first search tree, vertex 6.

5.11.4 Depth-first search algorithm

In a depth-first search, a vertex sequence F is maintained. As long as the vertex sequence F is not empty, the following steps are carried out in a loop:

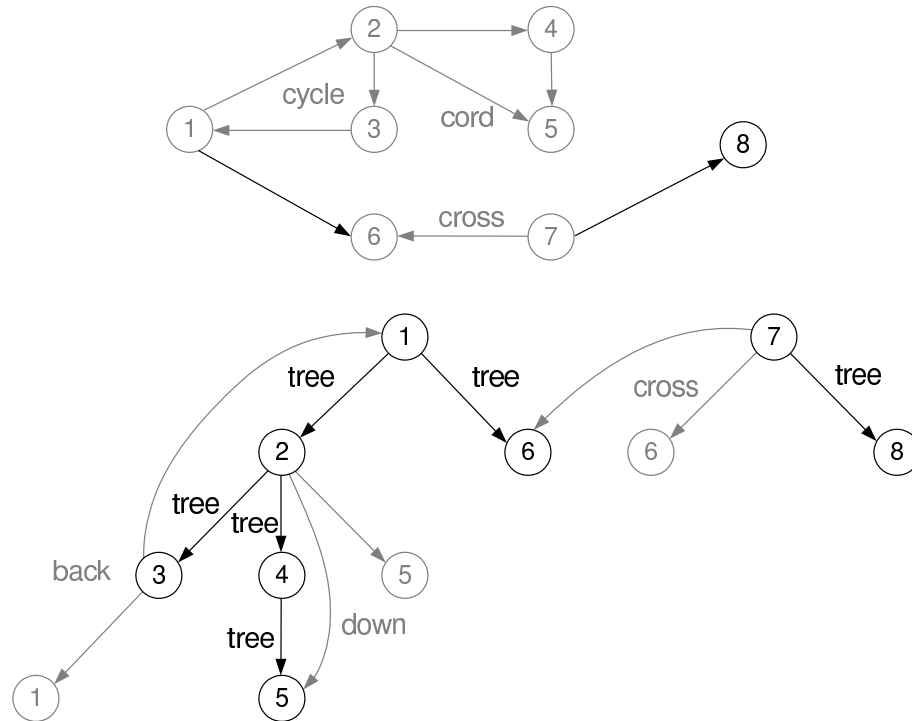


Figure 5.8: Edge classifications

- If the vertex at the end of F has a successor which has not been visited yet, such a successor is appended to the end of the sequence F .
- If the vertex at the end of F has no successor which has not been visited yet, it is removed from the sequence F .

The vertices visited and the edges used in the course of the depth-first search form the depth-first search tree. An unvisited vertex is chosen as the start vertex. If all the vertices have not been visited at the end of the process, a remaining unvisited vertex is chosen and a new vertex sequence is maintained. The process is repeated until all the vertices have been visited. A depth-first search tree is formed for each sequence. The depth-first search trees forms a depth-first search forest.

5.11.5 Depth-first search example

The graph in Figure 5.9 will be used to demonstrate a depth-first search.

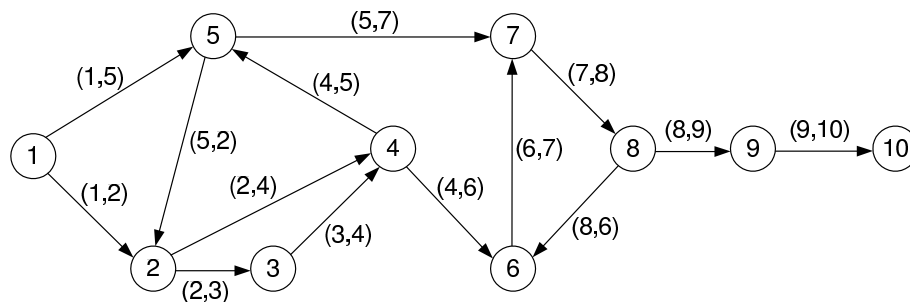


Figure 5.9: Graph example

The directed graph consists of ten vertices, labelled $1, \dots, 10$ and 14 edges, labelled $(1, 2), (1, 5), (2, 3), \dots$

Vertex 7 is chosen as the first unvisited vertex, giving it a pre-order number of 1. Vertex 7 has only one successor, vertex 8. Vertex 8 is still unvisited and is chosen as the next unvisited vertex. It is given a

pre-order number of 2 and the edge (7, 8) is classified as a tree edge. Vertex 8 has two successors, vertices 6 and 9. Vertex 6 is randomly chosen as the next unvisited vertex and given a pre-order number of 3 and edge (8, 6) classified as a tree edge. Vertex 9 will be considered at a later stadium. The successors of vertex 6 will be considered first. Vertex 6 has only one successor, vertex 7, which has been visited previously. Vertex 7 still has no post-order number, which indicates it as an ancestor of vertex 6. Therefore, edge (6, 7) is classified as a back edge. The presence of a back edge is an indication of a cycle. Therefore, the directed graph under consideration is not a directed acyclic graph. Since vertex 6 has no other successors, we leave it, giving it a post-order number of 1. Vertex 9, the remaining successor of vertex 8, is considered next.

After vertex 7 and all its ancestors had been processed, a new random unvisited vertex, vertex 2, is chosen. Vertex 2 is the root of the second tree in the depth-first search forest. After the depth-first search has been completed, the pre-order and post-order numbers shown in Table 5.3 were given to the vertices.

Table 5.3: Pre-order and post-order numbers for example 1

	1	2	3	4	5	6	7	8	9	10
pre-order no.	10	6	7	8	9	3	1	2	4	5
post-order no.	10	9	8	7	6	1	5	4	3	2

The depth-first search forest, edge classifications, as well as the search path, can be seen in Figure 5.10.

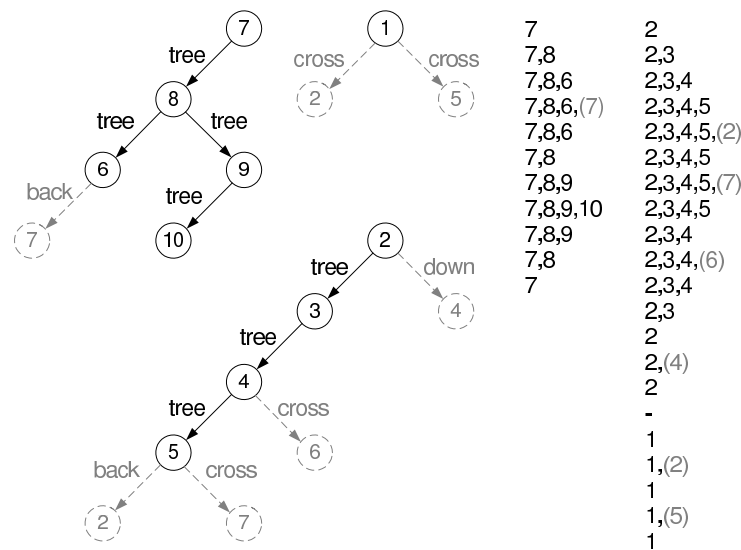


Figure 5.10: *Depth-first-search-forest-example-1*

The black vertices and edges indicate the depth-first trees. The vertices and edges in broken lines are not a part of the depth-first search trees and are only indicated to display the detection of back, down and cross edges. The first tree consists of vertices 6, 7, 8, 9 and 10. The second tree consists of vertices 2, 3, 4 and 5, while the third tree consists only of one vertex, vertex 1.

One of the many other depth-first search forests for the graph and its search path is shown in Figure 5.11. The pre-order and post-order numbers for this search shown in Table 5.4.

Table 5.4: Pre-order and post-order numbers of example 2

	1	2	3	4	5	6	7	8	9	10
pre-order no.	6	7	8	9	10	3	1	2	4	5
post-order no.	10	9	8	7	6	1	5	4	3	2

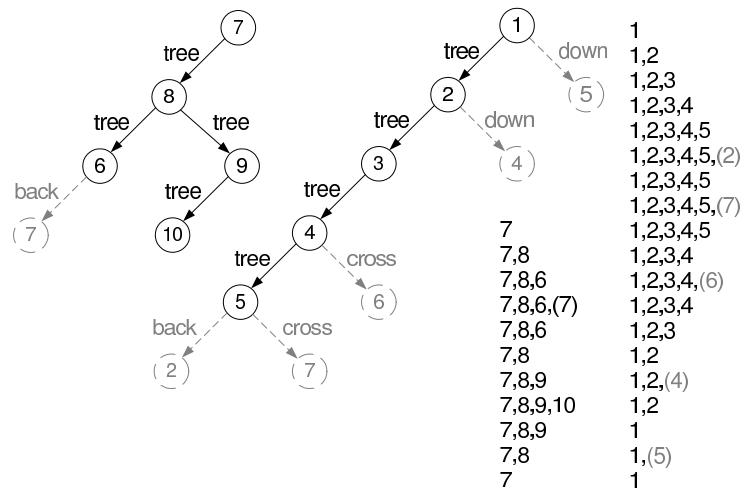


Figure 5.11: *Depth-first-search-forest-example-2*

The first tree consists of vertices 6, 7, 8, 9 and 10, the second tree consists of vertices 1, 2, 3, 4 and 5.

5.12 MATLAB implementation of basic graph functionality

Refer to Appendix D for the MATLAB code listing of basic graph functions with examples.

Chapter 6

Systems Theory

6.1 Introduction

This chapter provides a short overview of systems concepts and terminology and then defines system structure, function and behaviour suited to business system and business management system modelling.

Typical types of systems with some associated systems classification schemes are discussed.

For practical purposes the term system and model as well as systems model can be regarded as synonyms in the description given here.

A formal mathematical system structure definition and a sample mathematical model for system function and behaviour is provided. A short example of the application of this approach is given.

Activities in and around systems such as system analysis and design are explained.

Structural and functional aspects of business enterprise systems and business management systems are discussed. The measurement of business systems performance, leading to the control of the systems deployed in a business environment, is an important part of business management.

6.2 Systems concepts and terminology

The study of systems is referred to as systems thinking as well as ‘the systems approach’.

Cybernetics is a term used to encompass the science of systems. According to Wikipedia [126] it is an earlier but still-used generic term for many of the subject matter that are increasingly subject to specialisation under the headings of adaptive systems, artificial intelligence, complex systems, complexity theory, control systems, decision support systems, dynamical systems, information theory, learning organisations, mathematical systems theory, operations research, simulation, and systems engineering.

A more philosophical definition, suggested in 1956 by Frenchman Louis Couffignal (1902-1966), one of the pioneers of cybernetics, characterises cybernetics as ‘the art of ensuring the efficiency of action’.

Systems engineering and systems analysis are specialist disciplines devoted to the study of and development of systems.

6.2.1 System structure

A system may be defined as a set of elements in interrelation among themselves and with the environment. Von Bertalanffy [124].

Beishon and Technology Foundation Course Team [14], defines a system as a collection of parts or entities joined together in some organised way. The parts (known as components) are joined together in a logical organised way or in a random unorganised way.

The system can thus be seen as a set of interconnected elements / parts / components isolated for consideration by a human being.

A system displays structure as relationships exist among the components.

6.2.1.1 The system and its boundary

The system boundary indicates which elements are included in the system and which elements are excluded from the system. Boundaries are typically defined at discontinuities between groups of system components and processes (the definition of a process follows later) i.e. time, space, technology. System interfaces are found at the system boundary.

6.2.1.2 System components and system component attributes

System components or system component objects are the constituents or building blocks of systems.

System components can also be referred to as system elements or parts. The term component is however generally used.

Components can be grouped as system internal components or system boundary components.

Component classes are groupings of components into selected types. In specific systems components can belong to specified classes. An example of component classes used is classes such as monitor, comparator and reactor identified in instrumentation systems.

Components can have attributes which describe selected aspects of the component structure or behaviour. These attributes are used to capture the system status or state.

System component attributes can also be termed component attributes or component variables. The term attribute will be used here.

Component models are representations of system components using a specific modelling language with the associated syntax and semantics of the modelling language e.g. the unified modelling language (UML). See e.g. Booch et al. [18] or Alhir [3].

Does a null system i.e. a system with no components / parts exist? When using systems which adapt i.e. which gain or loose components it could be useful to define a null system i.e. a system without any components.

6.2.1.3 System component links

Links define the connections between system components as well as between systems. These links can be defined as relations between systems and system components. These links can be thought of as providing the *forces* between system components and systems.

6.2.1.4 System hierarchies

A hierarchy of systems can be used to identify subsystems and their behaviour. Recursion is possible where a system can consist of a number of components being systems themselves to the required level of complexity.

6.2.1.5 System attributes and properties

System properties can also be termed system attributes or system variables. These attributes are assigned to the system as a whole and can be used to store e.g. system state or performance data for the system. These attributes can also be seen as being special types of system components.

6.2.1.6 System component attributes and properties

Fixed component attributes are used to describe properties of the system attribute which do not change over time. Variable component attributes change over time. The variation can be of the discrete variable type or the continuous variable type. Limits and/or acceptable attribute values can be defined to ensure correct system operation.

6.2.1.7 Modifying the structure of a system

The system is changed when any of the parts are removed or a new part is added. The parts can also be modified when they are added to, or removed from the system. The term system structural configuration can be used to describe the structure of a system at a given time.

6.2.2 System function and behaviour

A typical system is doing something. As a model of a system evolves or operates component attributes are changed by the prescribed interactions among system components. The operational rules by which the components interact determine the system function.

6.2.2.1 System state

The system state represents the set of values for the system or system component variables one is concerned with at a particular moment in time. The system state can be defined as the status of the set of system components and system attributes at a given point in time for a given system structural configuration.

6.2.2.2 Capturing and modelling system behaviour

System behaviour constitutes the behaviour of the total system and can be identified in the movement or change of a system from one state to another. The behaviour is determined by a set of rules, typically defined in mathematical format, describing the interaction of selected system component attributes.

The set of initial component and system attributes are known as the initial or starting condition of the system.

The set of system and component attributes which evolve over time as the system operates constitute a record of the system trajectory.

A system can proceed along a system trajectory to reach a steady state - a process known as homeostasis. At a steady state the component attributes of the system remain at constant values.

As far as system behaviour and system processes are concerned it is possible to focus on behaviour of the total system or the working of a specific component or components of the system.

System growth is possible where components are added to the system as it operates - this is however reflected in the structural configuration information for a system.

6.2.2.3 System transformation and behaviour

With reference to system state a number of concepts can be defined.

System transformation is the specification of the changes of state occurring.

A system can have a single valued transformation i.e. going only to one state.

A closed transformation implies that no new states exist in the transformed system i.e. the system is reverting to a previous state.

The system trajectory can be formulated as a mathematical function on state variables. Refer to the basic example in section 6.4.3.

6.2.2.4 Classification of system functions

System functions can be classified according to a number of schemes.

A scheme described by Van Wyk [123] refers to technological elements in the form of matter, energy and information with a

- Storage function
- Manufacturing and processing function
- Transport and transfer function

for each of these technological elements.

A function which can be added here is a messaging function for information transfer.

6.2.2.5 Modifying the operation and function of a system

Properties of a system can emerge as the system is being studied or modelled. ‘The whole is greater than the sum of its parts’ - the concept that new properties emerge from a system arises from the ignorance of the analyst of the basic properties of the components of the system in the first place. According to a theorem stated by Ross Ashby [7], ‘The whole made by joining the parts is richer in ways of behaving than the system obtained by leaving the parts isolated.’

To modify the operation of a system the operational rules and or component attributes need to be modified. If the structure of a system is modified, modification of its operation can follow implicitly.

The parts not joined can exist separately in just as many total states as the joined system. The parts not joined cannot influence the changes from state to state of other parts while the joined parts can.

6.2.3 Control of systems

System control forms part of its behaviour but is treated separately here. The control of a system implies the implementation of system behaviour and special components and inputs based on the concepts listed below:

- Concept of goal setting and deciding on future states of a system
- Concept of goal setting and deciding on future required system performance measures to be attained.
- Concept of self regulation - a system controlling itself

It follows that the requirements for system control are:

- A goal state
- A system capable of achieving a goal state
- A means of influencing the behaviour of a system

To achieve system control a special subset of the system inputs (boundary components) need to be identified as control input. The system behaviour must be structured so as to make control of the behaviour possible via these control inputs. Monitoring of system outputs by humans or other systems providing control input feedback according to control rules is also required.

Three types of system controls can be identified:

- Open loop control:
This type of control depends on inputs remaining invariant or reasonably constant and on the relationship between control input setting and output value. An open-loop control system doesn't have or doesn't use feedback.
- Feedback control:
The current value of the output of the system is sensed by someone or something and the information obtained is used to make appropriate adjustments to the system inputs.
- Closed loop control:
The connection or link running between the input and output of the system forms a complete circuit linking the input and output of the system. Systems that utilise feedback are called closed-loop control systems. The feedback is used to make decisions about changes to the control signal that drives the plant. Refer to figure 6.1 taken from Barr [13].

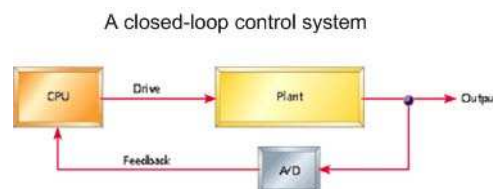


Figure 6.1: A system with a closed-loop control system

An important parameter which is of importance for controlling a system is system lag - the time delay between the moment of operation of the control and the effect appearing at the output of the system. Typical types of lags are transport lags, distance-velocity lags and exponential lags.

Self controlling systems, self organising systems, adaptive systems and artificial intelligence all represent systems where control is implemented as part of the system.

The messaging format used to control a system needs to be specified.

6.2.4 System performance measurement

System performance measurement requires the identification of relevant system component attributes to be monitored. These attributes can apply to internal or input and output system components.

The attribute values are then compared with goal attribute values, which allows system control to make adjustments to achieve performance as required.

System throughput and resource utilisation are typical system performance measures used.

Measuring system input and output to evaluate non-functional aspects of systems might also be required.

6.2.5 Types of systems

Systems can be typified according to their structure and function.

6.2.5.1 Systems typified according to component description

Pahl [91] identifies two general types of systems.

Models: A system whose elements are components is called a model.

Processes: A process is a system whose elements, called activities cause changes in the state of a product / artefact. Activities encapsulate changes in the state of artefacts or products which typically belong to a system distinct from the process the activities themselves belong to.

6.2.5.2 Systems typified according to structure

One can distinguish between open and closed systems according to the structure of systems.

Open systems have boundary components which interact with other systems or the system environment

Closed systems do not have boundary components.

6.2.5.3 Systems typified according to function

Systems can be typified as discrete or continuous, deterministic or probabilistic and as open or closed systems according to function.

Discrete systems - A discrete system is a system which can exist in one, and only one, of a certain number of clearly defined separate states at a time.

Continuous systems - A continuous system is a system which changes state in a continuous manner. By increasing the (time) intervals of state measurement or determination a continuous system can assume discrete changes of state / states.

Deterministic systems - A deterministic system is a system where the sequence of system states in the trajectory is predetermined.

Probabilistic systems - A probabilistic or stochastic system is a system where the exact sequence of system states cannot be predetermined. The behaviour of a system with probabilistic choice is a stochastic process.

Open systems - according to function - In an open system the end state can be reached via a number of different routes and from a number of different starting conditions. It can start of with one set of values and run to a final state and be started with another set of values and run to the same state.

Closed systems - according to function - In a closed system the end / final or equilibrium state can only be reached from one path. From a given set of initial conditions the system must follow one trajectory to the end state.

The *black box* is a special kind of system. For the black box only inputs and outputs are defined, control is possible and behaviour can be studied but no knowledge of the system components is available.

Table 6.1: System classification according to Kenneth Boulding [19]

<i>Level</i>	<i>Name</i>	<i>Description of level</i>
1	Static structure	Frameworks
2	Simple dynamic systems with predetermined necessary motions	Clockworks
3	Control mechanism or cybernetic system	Thermostat - self regulating in maintaining equilibrium
4	Open system or self maintaining structure	Level of the cell - life and non-life can be differentiated
5	Genetic societal	Plant - used in the empirical world of botany
6	Animal system	Animals have increased mobility, teleological behaviour and self-awareness
7	Human level or the human being as a system	The human system has self awareness and the ability to utilise language and symbolism
8	Social system or systems of human organisation	Considering: Content and meaning of messages Nature and dimensions of value systems Transcription of images into historical record Subtle symbolisations of art, music and poetry Complex complete set (gamut) of human emotions
9	Transcendental systems	Ultimates, absolutes, inescapable unknowables which exhibit structure and relationship

6.2.6 Classification of systems

Two examples of system classifications are given in tables 6.1 and 6.2.

6.3 System laws

Two *System Laws* have been formulated. These are the *Law of requisite system variety* and the *Principle of equifinality*.

Law of requisite system variety - To cope with the behaviour of a system one needs as least as much variety available as the system has. Refer to Ashby [7].

Principle of equifinality - There is not one best and only way to run an organisation to achieve a certain set of goals - many different routes can be taken to arrive at the same end state. This has implications for social systems and techno-social systems such as businesses as far as operation management is concerned. Refer to Ludwig von Bertalanffy [124].

6.4 Formal specification of systems

To provide a formal general system specification a mathematical approach is used here. Both the structural and functional aspects of a system need to be described in mathematical terms.

6.4.1 Formal system structure definition

This section provides a description of the basic general system suitable for business modelling. Equivalent terms used for system structure include system architecture and system anatomy.

The basic definitions proposed by Pahl [91] are used in the description below. A system has an environment and boundary and is defined as a domain containing seven sets i.e.

1. The set of system elements
2. The set of environment elements
3. The set of system boundary elements

Table 6.2: Systems classified according to mode of operation and the physical nature of their components and couplings; Jones and Edited by Singleton, W.T. et al. [66]

	<i>Kind of system and its mode of operation</i>	<i>Component</i>	<i>Couplings between components</i>	<i>Examples</i>
1	Manual system Operator directed, flexible	hand tools or aids	one human operator	cook plus utensils, craftsman plus tools, singer plus amplifying equipment
2	Mechanised system System directed, rigid	powered mechanical subsystems	on-line human operators, tracks, conduits, etc.	railway system, assembly line
3	Automatic system Pre-set, programmed or adaptive	powered mechanical subsystems	cables, pipes, conduits, levers, etc., forming a control circuit	clock, process plant, telephone exchange, digital computer
4	Collaborative man-machine system Exploratory and flexible	one or more human operators, one or more complete automatic systems	complex displays and controls	multiple-access computers
5	Mechanical subsystem Operator controlled and inflexible	highly interdependent physical parts forming indistinguishable components and couplings		engine, automobile, machine tool
6	Administrative system Goal directed and hierarchical	human operatives with tools or aids	rules, messages, human administrators and informal contacts	army of foot soldiers, a business, a school
7	Voluntary system Self-rewarding and collaborative	any number of persons each of whom is also a biological system and some of whom also act as administrative subsystems	affection, shared aims, laws, customs, managers, physical presence, mutual aid, common language, ancestry, etc.	family, religious order, club, society, (university?)
8	Environmental system Permissive of a range of human activities and contacts: prohibitive of others	inhabitants and facilities within an environment, the outside world	spaces and the barriers between and around the components	occupied building, city or region
9	Biological system Homeostatic, adaptive, evolutionary, growing, differentiating and self-reproducing	cells, organs, subsystems, all of which are also physical systems	nerves, glands, chromosomes, etc., past experience and environment	cells, plants, animals, human operators
10	Physical system Dynamically stable but subject to eventual decay	elementary particles, planets, seas, land, etc.	gravitation, electrical forces, radiation, physical motions and forces	solar system, molecule, crystal, cloud, strut, tie, beam, shell
11	Symbol system Semantic, analogous, ambiguous or precise	words, signs, symbols, numbers etc.	syntactical rules	languages, mathematics, codes, etc.

4. The property mapping of the elements to a set of literals
5. A relation or relations defined on the elements
6. An influence relation or relations defined on and linking the environment and system elements
7. An action relation or relations defined on and linking the system and environment elements
8. A model is defined as a system whose elements describe the state of a product (which can be an enterprise). A process is defined as a system whose elements describe changes in the state of a product. It is postulated that hierarchies of systems containing models and/or processes can be defined

Figure 6.2 contains a graphical representation of the formal system structure definition based on set concepts.

A system component element is an identified part of the universe of discourse. It is also referred to as a system component or system element in the discussion below.

A property or attribute is a mapping of a system element to a set of literals. Literals include integer values, real and complex values, characters and strings of characters as well as sets made up from the basic literals.

In set theoretic mathematical terms a system is a superset (or domain) consisting of seven sets. A *domain* is defined as a partially ordered set of sets which are consistent. The components of the system are elements which belong to sets as described in table 6.3. More detailed information on the definition is given in table 6.4.

Table 6.3: System set theoretic concept

<i>System component element</i>	<i>Description</i>
c_i	System internal elements belonging to set C
b_j	System boundary elements belonging to set B
e_k	System environment elements which interact with the system belonging to set E

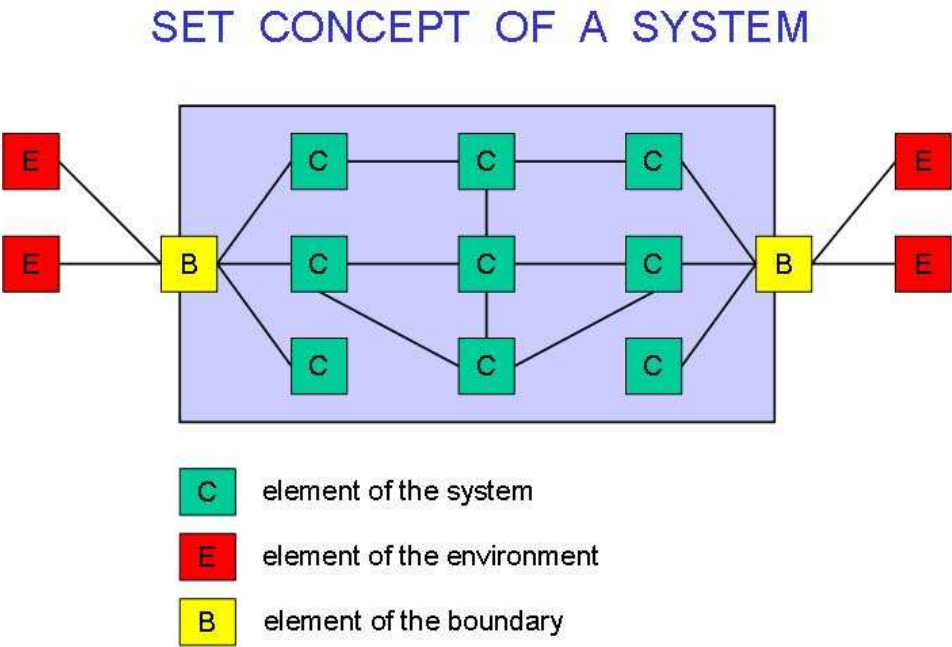


Figure 6.2: Graphical representation of a formal system structure definition (Pahl [91])

It is assumed that all interactions of the system with the environment occurs via the boundary elements of the system.

The system state is the collection of all the properties of the system at a given point in time.

System components which are literals can be defined and used to define the system state and other parameters which can be used to indicate system performance parameters.

The system boundary elements are defined to identify and specify the interaction between the system and its environment.

Table 6.4: Set theoretic system definition

<i>Item</i>	<i>Symbol</i>	<i>Description</i>
Internal components	$Set\ C = \{c_i\}$	Set of system internal component elements
Boundary components	$Set\ B = \{b_i\}$	Set of system boundary component elements
Environment	$Set\ E = \{e_i\}$	Set of component elements belonging to the system environment
Component Property mapping U_C	$U_C : C \rightarrow P_C$	A mapping of an element to a set of literals P_C the properties of the elements in set C
Boundary component Property mapping U_B	$U_B : B \rightarrow P_B$	A mapping of an element to a set of literals P_B the properties of the elements in set B
Internal Links	$R_C \subseteq C \times C$	A relation R_C defining links between system internal component elements in the form of ordered pairs of elements of the system which satisfies a specified condition
Internal-Boundary links	$R_B \subseteq B \times C$	A relation R defining links between system boundary and internal component elements in the form of ordered pairs of elements of the system which satisfies a specified condition
Other links	$B \times B$ and $C \times B$ and $C \times E$	Special links can also be defined
Influence	$F \subseteq E \times B$	A relation F defining links between the environment components and system boundary components in the form of ordered pairs of elements (environment, system boundary)
Other influences	$B \times E$ and $B \times B$	Special influences can also be defined
Action	$A \subseteq B \times E$	A relation A defining links between system boundary components and the environment components in the form of ordered pairs of elements (system, environment)
System	$S = \{C, E, B, U, R, F, A\}$	A superset (domain) containing seven sets $U = U_C$ and U_B
The set of system components	$C_S = C \cup B$	All system components

For the sake of completeness a null system, i.e. a system without any components, and a universal system containing all the components included in a given environment viewed as a system can also be defined.

System modification can be defined as the process of adding or removing components with the associated links, actions and influences to or from a system. A requirement that the system as a domain remains consistent must be added here. In principle one can start with a null system and add components with the associated relations and build upon a system in this way.

A system hierarchy consisting of sub- and super systems can be defined.

6.4.2 Formal system function definition

In describing system behaviour and system processes one can focus on the behaviour of the total system or on the working of each component of the system.

To account for system functionality a number of requirements need to be met. Aspects which need to be defined and dealt with are:

- System state
- Internal system component links and interactions
- System – environment links and interactions
- System control

- Interfaces between boundary components and system environment components
- Initial or starting conditions
- The system trajectory

The formal functional system definition given here is taken from Chapman, Bahill and Wymore [22]. The system function is described by three sets and the next state mapping as shown in table 6.5.

Table 6.5: System function formal specification

<i>Symbol</i>	<i>Description</i>
I_s	Set of system inputs
O_s	Set of system outputs or readouts
S_s	Set of system states
$R_s \subseteq S_s \times O_s$	System operational functional specification providing a relation between S_s and O_s
R_{ci}	The system structural relation linking components $i = 1, 2, 3, 4 \dots$
$N_s \subseteq \{(S_s \times I_s) \times S_s\}$	The next state mapping describing the logic of the sequence of states of the system
U	The set of system properties
T	System time/progress counters e.g. $T = \{0, 1, 2, 3, 4, 5, 6, 7, \dots\}$
$f = T \times I_s$	The system trajectory - mapping a discrete set of counters (time points) to I_s

6.4.3 Basic example of formal system structure and function modelling

Consider a system consisting of a light switch which controls a light and is operated by an operator interacting with the switch by setting it on or off and reading out the status of the light as on or off as shown in figure 6.3.

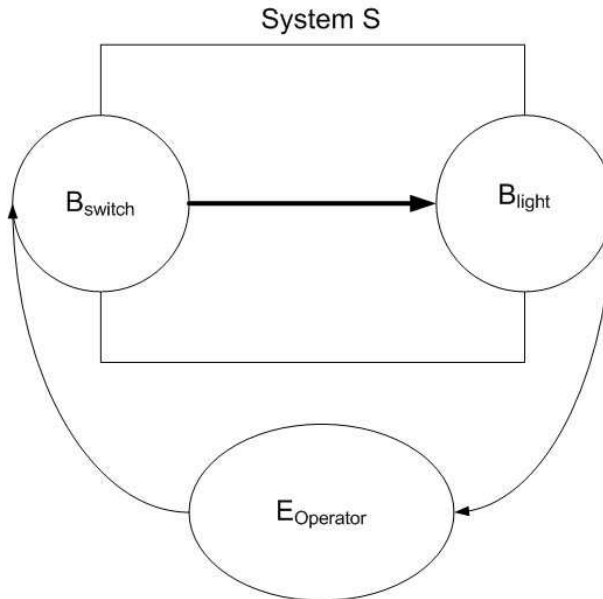


Figure 6.3: System Function Example

The sets used to describe the system shown in the figure are listed in table 6.6. All these relations can also be manipulated using boolean representations of the relations.

Table 6.6: System function example

<i>Set</i>	<i>Description</i>
$I_s = \{\text{setOn}, \text{setOff}\}$	Set of system inputs - Operator can set the switch to On or Off
$O_s = \{\text{lightOn}, \text{lightOff}\}$	Set of system outputs or readouts
$S_s = \{\text{on}, \text{off}\}$	Set of system states - The positions of the switch
$R_s \subseteq S_s \times O_s$ $R_s = \{(\text{on}, \text{lightOn}), (\text{off}, \text{lightOff})\}$	System operational functional specification providing a relation between S_s and O_s
R_{ci} $R_{control} = \{B_{switch}\} \times \{B_{light}\}$ $R_{control} = \{(B_{switch}, B_{light})\}$ and: $R_{operate} = \{(E_{operator}, B_{switch})\}$ and: $R_{sense} = \{(B_{light}, E_{operator})\}$	The system structural relation linking components Boundary system components B_{switch} and B_{light} are logically linked to define system operation. The influence relation $R_{operate}$ defines the logical link between the system environment component $E_{operator}$ and the system boundary component B_{switch} . The action relation R_{sense} defines the logical link between the system boundary component B_{light} and the component system environment component $E_{operator}$
$N_s\{(S_s \times I_s) \times S_s\}$ $N_s\{(\text{on}, \text{setOn}), (\text{off}, \text{setOff}),$ $(\text{on}, \text{setOff}), (\text{off}, \text{setOff})\}$ $\times \{\text{on}, \text{off}\}$ $N_s = \{((\text{on}, \text{setOn}), \text{on}),$ $((\text{off}, \text{setOff}), \text{off}),$ $((\text{on}, \text{setOff}), \text{off}),$ $((\text{off}, \text{setOn}), \text{on})\}$	The next state mapping describing the logic of the sequence of states of the system
$U_1 : B_{switch} \rightarrow \{\text{on}, \text{off}\} = S_s$ $U_2 : B_{light} \rightarrow \{\text{lightOn}, \text{lightOff}\} = O_s$ $U_3 : B_{switch} \rightarrow \{\text{setOn}, \text{setOff}\} = I_s$	The set of system properties $U = U_1 \cup U_2 \cup U_3$ The components of the system are mapped to their applicable sets of properties or attributes
$T = 0, 1, 2, 3, 4, 5, 6, 7$	System time/progress counters e.g. $T = 0, 1, 2, 3, 4, 5, 6, 7, \dots$ the counter 0 corresponds to the system initial state
$f \subseteq T \times I_s$ $f = \{(1, \text{setOn}),$ $(2, \text{setOff}),$ $(3, \text{setOn}),$ $(4, \text{setOn}),$ $(5, \text{setOff}),$ $(6, \text{setOff}),$ $(7, \text{setOn})\}$	The system trajectory - mapping a discrete set of counters (time points) to I_s
$N_s \circ R_s$ $f \circ (I_s \times O_s)$	Compositions - The composition $N_s \circ R_s$ supplies the result readout contained in O_s for the next state mapping; $f \circ (I_s \times O_s)$ provides the output trajectory of the system

6.5 MATLAB implementation of basic system function example

Refer to Appendix E for the MATLAB implementation of the basic system function example.

6.6 System analysis

A short summary of system analysis based on Beishon and Technology Foundation Course Team [14] is set out below. A more comprehensive treatment can be found in the book on Systems Engineering by Erik Aslaksen and Rod Belcher [8].

Systems analysis is the process that leads to a system specification. It supplies the engineering basis for system design.

According to Beishon [14] one needs to:

- Identify system, subsystems, boundaries and interrelations.
- Proceed inwards to lower levels of systems or outwards to the environment
- Postulate system functions.
- Identify feedback loops outside the system.
- Describe system behaviour – scientific analysis of phenomena needs to be done or be available.
- Formulate a quantitative connection among variables of a system.

When the behaviour can be predicted the system analysis should at a suitable level of detail. Aslaksen lists the following structure for systems analysis, [8].

- Information gathering
- High level modelling
 - Input and output identification
 - Types of models
 - Specifications as models
- System specification

6.7 System design

System design is the engineering activity for systems. The specification of a system which needs to be developed or built is planned and designed to the required levels of detail.

Typical activities of system design include:

- Set objectives, goals and aims of the system design exercise
- Develop a preliminary specification of primary purpose and objectives of the system
- Define the tentative boundary of the system
- Define subsystems and interrelationships
- Model the interaction among the subsystems
- Analyse subsystems to identify sub-subsystems if necessary
- Outline a functional model
- Define / recognise functional activities (primary and secondary)
- Define subsystem boundaries
- Construct or realise the system
- Test the system

With the aid of electronic computation devices (e.g. stored program digital computers) which enhance the mental ability of humans, it has become possible to study and develop increasingly complex systems.

6.8 Business enterprises and business enterprise systems

Emery and Trist [36] state that enterprises are better understood as open socio-technical systems which exhibit equifinality i.e. can reach their goals by many different routes. The system approach applied to a business enterprise reveals the hierarchical nature of the subsystems within subsystems, and draws attention to the need to provide control within the subsystems and between systems.

With open business systems there is also a need to control the relation of flow of goods, materials, money, information (MEI trilogy - Van Wyk [123]) five M's (men, materials, machines, money and management) across the interface between the system (the business) and the relevant environment.

The introduction of a systems model of a business can ensure effective and efficient business structure and processes as well as effective and efficient management structures and processes for a business. The term effective here indicates that the 'right' - appropriate things are being done and efficient indicates that processes are being done in the 'right' way.

The management of a business system is modelled as a separate system linked to the business system. The management system provides the necessary monitoring and control functionality to form the business as whole.

Figure 15.1 shows the interaction of business and management systems.

6.8.1 Business systems - structural models

The service business is a collection of persons, supporting infrastructure, knowledge and management organised to engage clients and to deliver services to selected clients.

The components of a business can be seen as artefacts i.e. products or objects made by humans. The artefacts can be physical or logical / conceptual in nature. Logical artefacts include activities and processes.

The business can be viewed as made up of physical and logical components which can be grouped into a number of subsystems. A given component can be contained in a subsystem of a system with a given logic and can also be contained in a subsystem of another system serving a complementary logic. A component should not be included in more than once in a system with a specific logic.

Huhnt [59] defines projects, processes, business processes, management processes, activities and functions as follows:

A project can be defined as a structured set of time consuming activities to be performed with a well defined start and well defined end.

A system whose elements are activities is called a process. Activities encapsulate and imply changes in the state of physical and logical artefacts and products. A process is a system whose elements describe changes in the state of a physical or logical product or artefact.

A process in the context of this development is a business process. A business process can be defined as a collection of related structured activities.

The term management process is used here as the collection of processes used to monitor and control business processes to achieve well defined goals.

A function can be defined as a task or activity which supports one or more corporate goals.

An activity can be defined as any task, job or operation which needs to be performed to complete a work package or project, Burke [21]. In the project management environment an activity needs to be completed in a specified time and uses a set of definable resources.

Business system attributes need to be defined on component, subsystem and system level to meet requirements set out for management monitoring and decision making.

Business system and subsystem boundaries are defined to delineate discontinuities in time and space as well as structural logic of interacting systems.

Business system boundaries for a service business are found where system components interact with the business environment for the exchange of goods and services and information as well as management. Interaction can be on the physical level such as interactions between persons in offices, machines passing information as well as management interaction. On the logical level boundaries can be defined between e.g. business functions controlled by the business and functions outsourced from other businesses.

A business has boundaries with and can form part of other systems such as inter business joint ventures, the regional economy, the national economy and so on.

A system whose elements are components is called a model. Business objects can be identified to serve as the components of business system models.

Three possible approaches to model business system components or objects are discussed below. Any physical or logical business component / entity can be grouped into a selected specified subsystem logic described below.

6.8.1.1 Physical business system components and subsystems

Physical business components include personnel, offices and laboratories with the associated energy supply, telecommunication links, environmental control, office and other technical equipment as well as physical documents, document storage systems and digital electronic information processing and storage systems. Transport provided for personnel can also form part of the physical business infrastructure. Cleaning and maintenance supplies and equipment can also be seen as part of physical business components. The MEI trilogy described by van Wyk [123] can be used to provide a classification here.

6.8.1.2 Logical and organisational business system subsystems and components

Logical groupings of physical business components are made to enhance the business processes and aid the organisation of the management of the business. Organisational groupings such as offices grouped per region, province and country, technical departments within offices or across office boundaries, technical groups of persons, ad hoc and longer term project groupings of persons can be identified.

According to Robertson [104] a three dimensional view of an organisation along the project, office and department axes is suitable for modelling engineering service type businesses. Departments are linked to technical disciplines and can span multiple physical offices.

6.8.1.3 Functional business subsystems and system components

Businesses can be viewed along functional lines for management or other purposes.

A typical functional grouping of subsystems which is useful in the service business environment includes functions such as:

- strategy, policy and long term planning
- marketing, promotion and public relations management
- finance, bookkeeping and auditing
- personnel
- facilities
- knowledge
- logistics
- production
- risk
- administration

Any physical or logical business component defined can be linked to the functional subsystems listed above.

Identification of management system functionality providing support for management are dealt with in part IV of this document.

6.8.2 The production process

The production processes in an engineering planning and design office provides a professional service to clients which is typically organised along project lines.

The project work needs to be supported by a number of information systems which can be integrated to various degrees. Information systems need to generate, communicate and store information in the required formats to be accessible for project work.

Chapter 11 contains a detail development of the Engineering process model which can be applied to the systems modelling and management of the production process in the professional engineering services enterprise.

6.8.3 Business systems supporting business operations

Systems supporting a business office facility includes logistical support, maintenance and upgrading / adaptation for *inter alia*:

- structural components of the building
- space division and utilisation partitions
- office furniture
- desktop equipment
- water supply waste water removal, irrigation and storm water drainage networks
- air supply and environmental control, air conditioning and ventilation ducts, heating and cooling systems
- lighting
- physical security and alarm systems
- energy supply
- office stationery supplies
- waste disposal and recycling
- transport units, transport hubs and terminal access
- telephony and information transfer, telecommunication cabling and equipment

The facilities management approach, which deals with these aspects in more detail, is treated further in chapter 21.

6.8.4 Business management systems - structural models

A techno-social system is a system formed by the combination of technological systems with human management systems. In the case of the services enterprise the typical system deployed involves a coupling of information and communication technology (ICT) systems and human management systems.

An engineering planning and design office provides a professional service to clients. The management processes in such an office is typically supported by a number of information systems which can be integrated to various degrees. Figure 18.3 shows an overview in system diagram format of an engineering office management system.

The management information systems used in engineering offices can be broadly classified as having either a project focus or a business enterprise focus.

Management information systems with a project focus

Systems which focus on engineering project work management include functionality to:

- structure projects in work packages, (work breakdown structure)
- define rates and unit costs
- budget project work
- record personnel time
- record disbursements
- manage project related creditors
- manage project related invoicing and debtors

Manpower skill and knowledge data bases can also be linked to these systems.

Reporting functionality includes time, expense and cost, work in progress, cash flow, project task progress, earned value, project invoicing and reporting of creditor and debtor information.

Engineering office production management information systems reflect the flow of production units of work and information in the organisation.

Management information systems with a business enterprise focus

To provide management support for ongoing business activities which are not directly project related information systems which focus on business enterprise management need to be in place. Systems which focus on business enterprise management and provide management support are:

- scenario modelling system for long term planning - e.g. morphological modelling
- strategy and policy formulation support
- company organisation modelling
- marketing and business contact management
- personnel management
- personnel remuneration, salaries and time rates
- production project and work in process management
- logistics management system for office supplies
- financial budgeting, business cash flow, and accounting systems
- financial management
- office facility management
- asset registers
- knowledge management
- archiving, document and library management
- risk management
- communication monitoring and control
- general administration

To be useful, management systems need to support formalisation of business goals, identify and quantify inputs, outputs and specify business system control links.

Engineering office business management information systems reflect the structure of the underlying datasets used to model the organisation and its resources.

When modifications are made to any of these structures, system reference data (meta data) needs to be updated to maintain the required system functionality.

This updating operation presently not well supported in commercially available software. Functionality to accommodate changes to business and workflow models, on which systems base data processing and management reports and to apply these in a controllable way to these systems is required. These changes can impact on the format and validity of historic and live data. Maintaining system integrity requires robust system structures and functionality which need to be researched and studied.

6.9 Formulation of required business model structure and functionality

In this section an outline of the application of systems theory to the modelling of a professional service business, like the professional engineering practice is supplied. Businesses are modelled using business objects and business processes which are controlled by management processes, which in turn can refer to management objects.

Models of business objects used in information processing typically relate to a number of logical classification structures which are used to view the business and associated organisation when communicating information about the business. In this section typical business object classification structures are reviewed which are used to organise data about business objects. The functionality required in the use of these hierarchies is set out and an abstract concept which can lead to a software implementation with the required form and function, is described.

6.9.1 Typical business object classification structures

A business is viewed as a collection of business objects which can be grouped into artefacts, personnel, activities and information (monetary aspects are included under information).

This is similar to Huhnt [61], which uses a collection of persons, tasks, tools and data in the grouping of artefacts which is modelled to support project planning.

In the process of structuring the business to make it functional and allow for its management, classification hierarchies are defined and developed to link business objects to. These hierarchies may relate to physical or abstract attributes/aspects of the business objects. All relationships between business objects are defined in terms of these hierarchies.

Examples of classification hierarchies are set out in table 6.7.

To be able to report on attributes of selected business activities one needs to be able to select objects based on typical classification hierarchies defined in table 6.7 and generate applicable datasets for review using the functionality described below.

6.9.2 Conceptual model of system functionality to process business objects

Each business object will need to be connected to zero or more nodes in a hierarchy. The functionality to couple/uncouple an object to a hierarchy node will be required.

Generation of attribute data of business objects based on selections made using said hierarchies and combination of hierarchies is typically needed to achieve management and other reporting output required for day to day running of the business.

New hierarchies may be added from time to time and complete hierarchies removed. The terms used in a hierarchy might also need to be changed from time to time. New elements or groups of elements may need to be added to existing hierarchies and elements may need to be removed from existing hierarchies. This implies a versioning requirement for hierarchies.

Attributes of objects may need to be subtotalled or grouped as hierarchies are traversed in the reporting and data generation process.

Software functionality for reporting resembling the ‘Pivot table’ found in spreadsheet applications might apply here.

To achieve the desired structure and functionality in a model of business objects it is postulated that the object hierarchies need to be defined as structured sets from which

Table 6.7: Classification hierarchies used in business organisation

<i>Classification</i>	<i>Typical Class Instances</i>
Technological (Van Wyk [123])	Matter, energy, information
Business artefacts/objects	Persons, tasks, tools, datasets
Physical / geographical	Offices grouped into divisions/ regions / provinces / countries, geographical co-ordinates latitude and longitude
Functional departments	Strategic planning, Marketing, production; project management, financial, purchasing and logistics, administration, personnel, after sales service, public relations (Du Plessis [32] and Du Plessis [33])
Legal company structure	Holding company with subsidiary companies
Major business goals	Marketing, production, quality control, after sales service
Business activities	Operating and services / head office divisions
Types of artefacts owned by the business	Furniture, vehicles, computers, office machines, telephones and telecommunication equipment typically used in asset management systems
Management activities	Investigating, estimating, forecasting, scanning the environment, planning, organising, controlling, co-ordinating and commanding De Villiers [28]
Accounting	Assets, liabilities, current assets, current liabilities, owners equity, income accounts, expense accounts
Personnel	Owners, directors, associate shareholders, other shareholders
Personnel education	Matriculants, Technicon Diplomates, University graduates
Personnel cultural qualities	Nationalities, race groups
Management structure	Executive director, office manager, department manager, financial manager
Tax and government levies	Income tax, regional service levies, value added tax, municipal property tax, municipal service tax, government training levies ...
Marketing and public relations	Government clients, private sector clients, corporate clients, mining houses ...
Time	Calendar dates
Time periods	Years, months, days, hours, accounting periods
Technical discipline and sub-discipline hierarchies	Structural engineering Municipal engineering Coastal engineering Marine engineering Water supply engineering Water treatment Software engineering

subsets can be derived using a tree like graph structure and as subsets contained in each other.

It might be required to also define weights for vertices in graph tree structures which are used to derive datasets when trees are traversed to generate information as required.

This approach is demonstrated in section 12.2.

Chapter 7

Relational Database Theory

7.1 Introduction

History of relational database theory

According to Wikipedia [126] the relational database model was invented by E. F. (Ted) Codd [24] as a general model of data, and subsequently maintained and developed by Chris Date and Hugh Darwen among others. In The Third Manifesto (first published in 1995) Date and Darwen show how the relational model can accommodate certain desired object-oriented features without compromising its fundamental principles. The foundation for the relational model was laid by important works published by Georg Cantor (1874) and D.L. Childs (1968). Cantor was a 19th century German mathematician who was the principal creator of set theory. Childs is an American mathematician whose ‘Description of a Set Theoretic Data Structure’ was cited by Codd in his seminal 1970 paper ‘A Relational Model of Data for Large Shared Data Banks’. Childs used set theory as the basis for querying data using set operations such as union, intersection, domain, range, restriction, cardinality and Cartesian product. The use of sets and set operations provided independence from physical data structures, a pioneering database concept at the time.

7.2 Database System Concepts and Architecture

A Database Management System (DBMS) is a collection of programs that enables users to create and maintain a database.

The architecture of Database Management Systems (DBMS) has evolved from the monolithic integrated system hosted on a mainframe computer to the client-server architecture. For the client-server architecture the client (user or developer) can use local area network (LAN) or wide area network (WAN) and internet access to the database hosted on a server machine. The *client* module typically runs on a user workstation which supports a Graphical User Interface (GUI) environment. A *server module* handles the data loading, access and manipulation functionality for the database. Refer to figure 7.1 which shows a typical logical system component overview.

In this section the concepts relating to the categories of data models, schemas, instances of a database, database architecture logic and modelling techniques such as the entity-relationship, object techniques and Unified Modelling (UML) are discussed.

7.2.1 Data Models, Schemas and Instances

The outcome of the logical design of a database is called a database schema which describes the database. Data modelling techniques have conventions for displaying schemas in diagram format.

Each object contained in the schema is referred to as a schema construct.

The data in a database changes as it is used. The data status at a specific moment is called the database state, the current set of occurrences or instances in the database.

When a database is defined the schema is specified using the appropriate DBMS functionality. At this point the database is not populated with data. The initial state of the database is reached when the data is loaded for the first time. When any updates or additions to the data in the database are made, the database moves to a new state. The DBMS should ensure that every state reached is a valid system state for the database and that the database integrity is maintained as it moves from state to state. This

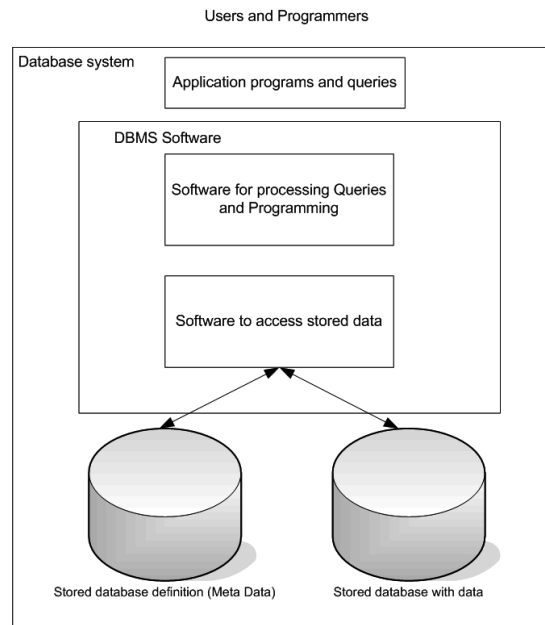


Figure 7.1: *Simplified database system environment logic*

can be achieved through a transaction mechanism. Refer to section 6.4.2 where analogous concepts are discussed in general systems terms.

The DBMS stores the schema definition in a database catalogue which can be seen as *meta data* i.e. data describing the structure and format of data. If the structure of the database needs to be modified a *schema evolution* needs to take place. The sophistication of DBMS software, and the detailed nature of the schema update required, determines to what extent schema updates can be done while a database is operational.

7.2.2 DBMS Architecture

Three important characteristics of the database approach are:

- insulation of programs from data,
- support of multiple users and multiple user views on the data,
- catalogue usage to store the database schema.

Three-Schema Architecture

In the three level schema or three-schema database architecture separation of the user applications from the physical database with stored data is achieved by defining and using schemas on three levels:

1. The *internal level* has an *internal schema* which defines the physical storage structure of the database using a physical data model.
2. The *conceptual level* has a *conceptual schema* which is a high level data model and describes the structure of the database to its users and programmers. The schema hides the details of physical storage structures and describes entities, data types, relations, user operations and constraints.
3. The *external* or *view level* is a high level data model and includes *external schemas* or *user views*. It describes the part of the database that a group of users are interested in and hides the rest of the database.

Refer to figure 7.2 which shows these concepts in diagrammatic form.

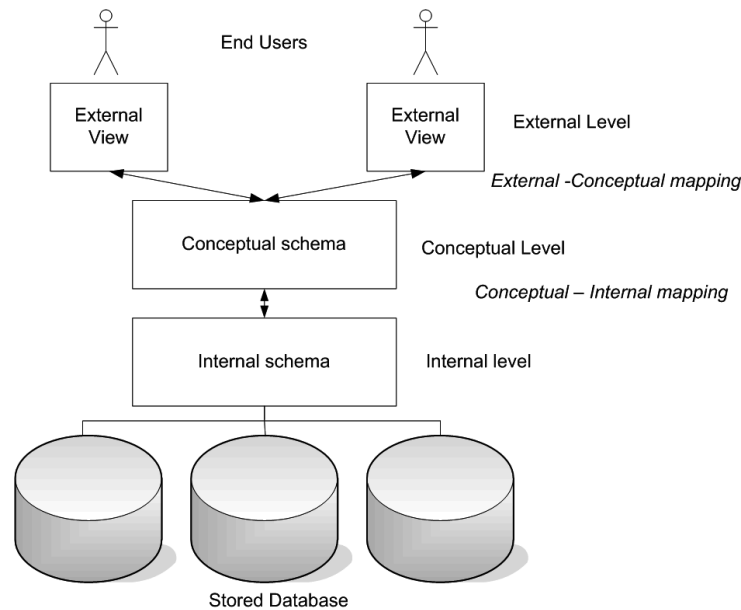


Figure 7.2: Three level schema database architecture

Data Independence

Data independence is defined as the capacity to change a database schema at one level without having to change the schema at a higher level. Two types of data dependence can be identified:

1. *Logical data independence* is the capacity to change the conceptual schema without having to change the external schema or database application programs.
2. *Physical data independence* is the capacity to change the internal schema without having to change the conceptual or external schemas.

7.2.3 Data Modelling Techniques

Various techniques have been developed to aid the database design process.

There are a number of database design diagramming techniques in use which aid the visual representation of the relational model. The Entity-relationship Diagram (ERD), and the related IDEF diagram used in the IDEF1X method conceived by the U.S. Air Force, based on the ERD, were originally developed.

The tree structure of data may enforce hierarchical model organisation, with a parent-child relationship table.

With the advent of object modelling Enhanced Entity Relationship (EER), Unified Modelling UML and Object Modelling Techniques (OMT) were added to the suite of techniques available to database designers.

The Entity-Relationship Model

The entity relationship (ER) notation is summarised in figure 7.3. (Google Images [45]). The ER model describes data using entities, relationships and attributes (entity properties) where:

- entities are real word objects or things
- relationships are attributes of one entity type referring to that of another entity type
- attributes are entity properties.

Further material on this technique can be found in Elmasri and Navathe [35] and Fertuck [39].

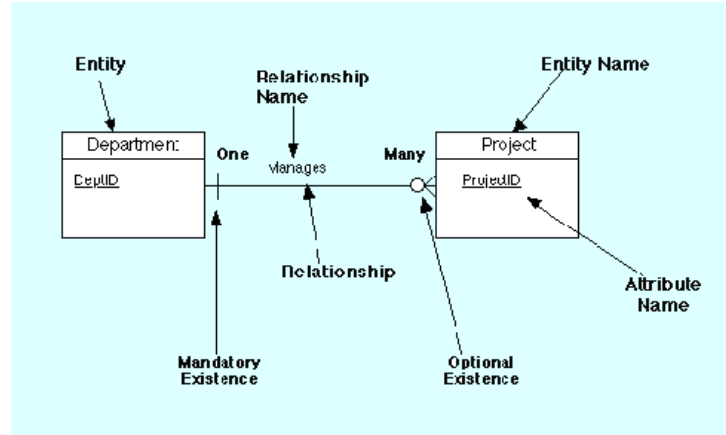


Figure 7.3: Entity Relationship Diagram Notation

Enhanced Entity Relationship Technique

The Enhanced Entity Relationship (EER) model includes all modelling functionality of the ER approach with the addition of object concepts such as *class* and *superclass* and also *specialisation* and *generalisation*. The mechanism of *attribute inheritance* and *relationship inheritance* is also added to the ER technique.

Specialisation is the process of defining a set of subclasses of an entity type while generalisation is the reverse process i.e. defining the superclasses which entity types belong to.

Object Modelling and the Unified Modelling Approach

The use of object modelling methodology such as the Unified Modelling Language (UML) and Object Modelling Techniques (OMT) is becoming more common.

UML class diagrams contain the object class displayed as a rectangle. Inside the rectangle sections that show the class name, attributes for the class or objects of the class and the methods (operations) which can be applied to the class or objects of the class are shown.

Figure 7.4 shows an example of a UML class diagram taken from Elmasri and Navathe [35].

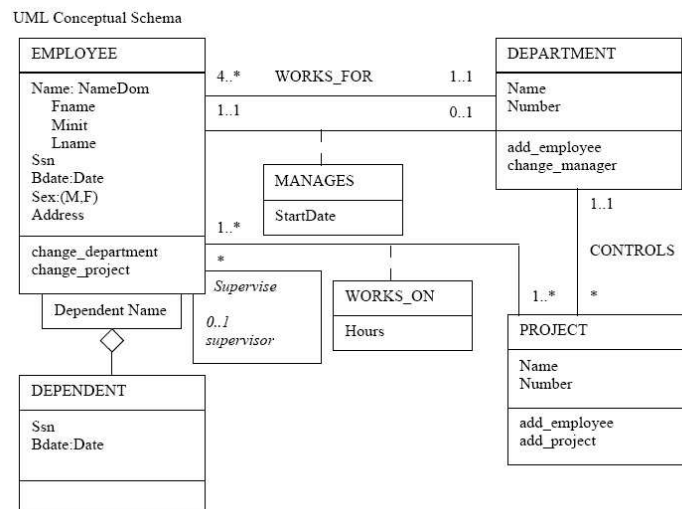


Figure 7.4: UML Conceptual Schema

More detail on the UML approach is contained in references such as Alhir [3], Gomaa [43] and Booch et al. [18].

7.3 Relational Data Model, Constraints and Relational Algebra

In the relational model a database schema is said to consist of a set of relation names, the headers that are associated with these names and the constraints that should hold for every instance of the database schema. The material described in this section refers to Elmasri and Navathe [35] and material contained in Wikipedia [126].

7.3.1 Relational Model Concepts

The relational model represents a database as a collection of relations. Each equivalence relation can be represented by a table of values where each row represents an ordered collection of data values. The table identifier (relation name) and column identifiers (attribute names) are used to interpret the meaning (semantics) of the data in the table. All values in a column have the same data type.

In the formal relational model definition a row in a table is termed a *tuple*, a column header an *attribute* and the table a *relation*. The domain of an attribute describes the logical description and data type of each column of the relation.

The formal definition of the concepts noted above follows.

Domains and Domain Constraints

A *domain* D is a set of *atomic* values. Atomic indicates that each value in the domain is indivisible as far as the relational model is concerned. A domain needs a logical definition and a *data type* with a *format* to be specified for it.

Examples of domains are:

- Employee_Number: A set of four digit integer numbers
- Employee_Surname: A set of alphabetic character strings making up acceptable surnames
- Hourly_Rate: A set of floating point values specified to two decimals to form monetary values
- Customer_Code: A set of 10 character alphanumeric identifiers for customers
- Customer_Name: A set of alphanumeric character strings making up acceptable customer names

When a domain is logically specified and a data type has been chosen for the domain it is then constrained.

Refer to section 7.12 for computer implementation aspects relating to domains and constraints on domains.

Relation schema

A *relation schema* R is denoted by $R(A_1, A_2, A_3, \dots, A_n)$ and consists of a relation name R and a list of attributes $A_1, A_2, A_3, \dots, A_n$.

For a given number of attributes $A_1, A_2, A_3 \dots A_n$ the set formed by all the attributes $R = \{A_1, A_2, A_3 \dots A_n\}$ is defined as a relational schema.

A relation schema describes a relation where R is the name of the relation.

A relation schema (H, C) consists of a header H and a predicate $C(R)$ that is defined for all relations R with header H . A relation satisfies the relation schema (H, C) if it has header H and satisfies C .

In predicate logic, a relational schema is also referred to as a relational *intension*.

Attributes

Each *attribute* A_i refers to the role played by some domain D in the relation schema R . The domain of A_i is denoted by $dom(A_i)$

Tuples

An *n-tuple* or *tuple* is a ordered list of n values $t = (v_1, v_2, v_3, \dots, v_n)$. Each value v_i with $1 \leq i \leq n$ is an element of $dom(A_i)$ or a special *null* value indicating that it is not specified. The i^{th} value in tuple t which corresponds to attribute A_i is referred to as $t(A_i)$.

A tuple is a partial function from attribute names to atomic values. A header is defined as a finite set of attribute names. The projection of a tuple t on a finite set of attributes A is $t[A] = (a, v) : (a, v) \in t, a \in A$.

Table or relation name

A_1	A_2	A_3	\dots	A_n	\leftarrow Attribute or column names
a_{11}	a_{12}	a_{13}	\dots	a_{1n}	
a_{21}	a_{22}	a_{23}	\dots	a_{2n}	\leftarrow Row or tuple values
a_{31}	a_{32}	a_{33}	\dots	a_{3n}	
\dots					
a_{n1}	a_{n2}	a_{n3}	\dots	a_{nn}	

Table 7.1: General format of a relation represented as a table

Relations or Relation states

The following definition of a relation formalises the contents of a table defined in the relational model:

A relation is a tuple (H, B) with H , the header, and B , the body, a set of tuples that all have the domain H . Such a relation closely corresponds to what is usually called the extension of a predicate in first-order logic except that here we identify the places in the predicate with attribute names.

A *relation* or *relational state* of the relational schema $R(A_1, A_2, A_3, \dots, A_n)$ is denoted by $r(R)$ is a set of n -tuples $r = \{t_1, t_2, t_3, \dots, t_n\}$. Refer to table 7.5 for an example of a relation.

The relational state is also referred to as a *relational extension*.

Degree, Cardinality and Cardinality Ratio of a Relation

The *degree* of a relation is the number of attributes in its relation schema n .

The *cardinality* of a domain D is the number of values in the domain and is denoted by $|D|$.

The *cardinality* or *cardinality ratio* of a binary relation specifies the number of relationship instances an entity can participate in. The possible cardinality ratios for a binary relationship are $1 : 1$, $1 : N$, $N : 1$ and $M : N$.

7.3.2 Tabular representation of a relation

A relation is represented in a relational database as a two-dimensional table where the table name is the relation name each column an attribute A_n or column name associated with the domain X_n and each row formed by the associated tuple values making up the relation. Refer to table 7.1

A specific instance of a relation is then the table described above with specific tuple values supplied.

7.3.3 Set theoretic Formulation of a Relational Database

The term relation (database relation) used in this chapter is based on the mathematical concept of a relation dealt with in Chapter 4, but is to be seen as a distinct specialisation based on the mathematical concept of a relation. The term relation used in this chapter refers to this specialisation, unless stated otherwise.

A relation is defined as a subset of the Cartesian product of the domains of the attributes of the relation. The relation and domain are the basic object types of the relational data model described by Elmasri and Navathe [35] and Greeff [48].

A relation $r(R)$ is defined on sets $X_1, X_2, X_3 \dots X_n$, where $X_1 = \text{dom}(A_1), X_2 = \text{dom}(A_2), X_3 = \text{dom}(A_3), \dots, X_n = \text{dom}(A_n)$ the domains of the attributes of the relation.

$$r(R) = \{(x_1, x_2, x_3 \dots x_n) \mid x_1 \in X_1, x_2 \in X_2, x_3 \in X_3, \dots x_n \in X_n\} \quad (7.3.1)$$

The sets $X_1, X_2, X_3 \dots X_n$ are the domains and $(x_1, x_2, x_3 \dots x_n)$ is a tuple which is an ordering of the variables $x_1, x_2, x_3 \dots x_n$ i.e. a generalisation of the ordered pair concept to n variables.

The relation $r(R)$ is a subset of the Cartesian product of all the domains as shown in equation 7.3.2.

$$\begin{aligned} r(R) &\subseteq \{X_1 \times X_2 \times X_3 \times X_4 \dots \times X_n\} \\ r(R) &\subseteq (\text{dom}(A_1) \times \text{dom}(A_2) \times \text{dom}(A_3) \times \dots \times \text{dom}(A_n)) \end{aligned} \quad (7.3.2)$$

The Cartesian product specifies all the combinations of values from the underlying domains.

If all the domains are finite the total number of tuples in the Cartesian product is given by equation 7.3.3

$$|dom(A_1)| * |dom(A_2)| * |dom(A_3)| * \dots * |dom(A_n)| \quad (7.3.3)$$

As an alternative a relation $r(R)$ can be defined as a finite set of mappings $r = \{t_1, t_2, t_3, \dots, t_n\}$ from the relational schema $R = \{A_1, A_2, A_3, \dots, A_n\}$ to the union of the domains of all the attribute domains $D = dom(A_1) \cup dom(A_2) \cup dom(A_3) \cup \dots \cup dom(A_n)$. Each tuple t_i is a mapping from R to D and $t(A_i) \subset dom(A_i)$ for $1 \leq i \leq n$.

7.3.4 The structural properties and characteristics of a relation

The structural properties of a relation are summarised in Dutka and Hanson [34, Figure 2.1, page 13] and Greeff [48].

1. Columns represent database fields. Each column has an unique name. Columns are also referred to as attributes.
2. Each column is homogeneous. The entries in a column are all of the same data type and format.
3. Each column entry is contained in the domain of the column i.e. the set of possible vales that the entry can contain.
4. Rows (tuples) represent records. If a relation as n columns, each row is an n-tuple.
5. The order of the rows (tuples) is not important.
6. The order of the columns is not important. Logically a column entry must however correspond to the attribute contained in that column.
7. All attribute values are atomic.
8. No duplicate rows are allowed.
9. Repeating attributes or groups of attributes i.e. collections of logically related attributes which occur more than once in a record occurrence are not allowed.
10. A *relational schema* is a set of attributes, dependencies and other constraints that characterises a relation. Various types of dependencies and constraints can be defined.
11. An *instance* of a relation is a set of rows that populate the relation. Updates to the database data can change the instance of a relation. An instance is valid if all the dependencies and constrains specified in the relational schema are satisfied.

7.4 Candidate keys and Primary key of a Relation

A key provides the basic mechanism for retrieving tuples within any table of a relational database.

A candidate key of a relation R with attributes $A_1, A_2, A_3 \dots A_n$ is defined as any subset K of the set $\{A_1, A_2, A_3 \dots A_n\}$.

$$K \subset \{A_1, A_2, A_3 \dots A_n\} \quad (7.4.1)$$

K must satisfy both the following conditions:

1. *Uniqueness property of a key:* For any two distinct tuples t_1 and t_2 of relation R there exists an attribute A_j of K such that $t_1(A_j) \neq t_2(A_j)$.
2. *Minimality property of a key:* $K' \subset K$ does not satisfy the uniqueness property. No element of K can be discarded without destroying the uniqueness property. The number of attributes that make up the key is a minimum.

A relation can have a number of candidate keys and one of the candidate keys is selected as the *primary key* of the relation. A relational database system only allows one primary key per table. *Single primary keys* are composed using one attribute while *composite primary keys* are made up using more than one attribute. The remaining candidate keys are referred to as *alternate keys*.

Since a primary key is used to uniquely define the tuples or rows of a relation the *integrity constraint* requires that none of its attributes may have a *null* instance. A null value represents missing data in a relation.

A *superkey* K of a relation is any a key formed such that $K \subset K'$. This is a set of attributes that contains the key.

The *entity integrity constraint* states that no primary key can assume a null value. If null keys were allowed it would not be possible to identify some tuples in a relation.

7.5 Prime Attributes

An attribute of an relation schema R is called a *prime attribute* if it is a member of some candidate key of R . An attribute is *nonprime* if it is not a prime attribute and is thus not a member of a candidate key. Attributes which are part of any key are called *prime attributes*.

7.6 Foreign Keys

The *referential integrity constraint* is specified between relations and states that a tuple in one relation referring to another relation must refer to an existing tuple in that relation. To formally define a referential integrity constraint the concept of a foreign key is required.

A foreign key is defined as follows:

If two relations R and S are defined on the same relational database the set of attributes F of relation R is said to be a *foreign key* of relation S with respect to S if both conditions below are satisfied.

- the attributes used to form F have the same underlying domain as a set of attributes of relation S that have been defined as the primary key of S .
- the values of F in any tuple belonging to R are either *null* or must appear in the primary key values of relation S .

This implies that tuples of relation R must refer to tuples of relation S that already exist. This is the *referential integrity* condition imposed on foreign keys.

7.6.1 Properties and characteristics of keys of relations

The properties of keys of relations are summarised in Dutka and Hanson [34, Figure 2.1, page 13] and Greeff [48].

1. A candidate key is an attribute, subset of the attributes or a function of and attribute or subset of the attributes that uniquely defines a row. A candidate key must have the following properties:
 - *Unique identification*: For every row, the value of the key must uniquely identify that row.
 - *Non-redundancy*: No attribute in the key can be discarded without destroying the property of unique identification.
2. A *primary key* is a candidate key selected as the unique identifier. Every relation must contain a primary key. The primary key is usually the key selected to identify a row when the database is physically implemented.
3. A *superkey* is any set of attributes that uniquely identifies a row. A superkey does not require the redundancy property. A superkey is written as a finite set of attribute names.
4. A *foreign key* is an attribute that appears as an non-key attribute in one relation and as a primary key attribute (or part of a primary key) in another relation.
5. A *composite key* is a key derived from more than one attribute.
6. All non-key attributes are functionally dependent on the key attributes. A set A of the attributes of a given relation R_i is functionally dependent on another set B of the attributes of R_i if and only if each value of B has associated with it one value of A in R_i at any given time.

7.7 Key constraints

One of the most important types of relation constraints is the *key constraint*. It tells us that in every instance of a certain relational schema the tuples can be identified by their values for certain attributes. A relation is defined as a set of tuples. Seeing that all elements of a set are distinct all tuples in a relation are distinct and no two tuples can have the same combination of values for all attributes. No rows are repeated in the tabular form of the relation.

A superkey (SK) is defined as a subset of the attributes of a relation. Thus for any two distinct tuples t_1 and t_2 in a relation state r of R the key constraint reads:

$$t_1(SK) \neq t_2(SK) \quad (7.7.1)$$

A superkey SK holds in a relation (H, B) if $SK \subseteq H$ and there are no two distinct tuples t_1 and t_2 in B such that $t_1(K) = t_2(K)$.

A superkey SK holds in a relation universe U over a header H if it holds in all relations in U . A superkey K holds as a candidate key for a relation universe U over H if it holds as a superkey for U and there is no proper subset of SK that also holds as a superkey for U .

A key is determined from the meaning of the attributes. Its value is based on the properties of the attributes. A key needs to be time invariant and must hold when new tuples are inserted into a relation.

7.8 Functional Dependencies in Relations of Relational Databases

A functional dependency (FD) is written as $X \rightarrow Y$ with X and Y finite sets of attribute names.

A functional dependency $X \rightarrow Y$ holds in a relation (H, B) if X and Y are subsets of H and for all tuples t_1 and t_2 in B it holds that if $t_1(X) = t_2(X)$ then $t_1(Y) = t_2(Y)$.

A functional dependency $X \rightarrow Y$ holds in a relation universe U over a header H if it holds in all relations in U .

A functional dependency is trivial under a header H if it holds in all relation universes over H . It can be proved that a FD $X \rightarrow Y$ is trivial under a header H if and only if $Y \subseteq X \subseteq H$.

It can also be proved that a superkey SK holds in a relation universe U over H if and only if $K \subseteq H$ and $K \rightarrow H$ holds in U .

Let S be a set of functional dependencies. The closure of S under a header H , written as $S+$, is the smallest superset of S such that:

- (reflexivity) if $Y \subseteq X \subseteq H$ then $X \rightarrow Y$ in $S+$
- (transitivity) if $X \rightarrow Y$ in $S+$ and $Y \rightarrow Z$ in $S+$ then $X \rightarrow Z$ in $S+$
- (augmentation) if $X \rightarrow Y$ in $S+$ and $Z \subseteq H$ then $X \cup Z \rightarrow Y \cup Z$ in $S+$

These statements are referred to as Armstrong's rules [6].

It can be proved that Armstrong's rules are sound and complete, i.e., given a header H and a set S of FD's that only contain subsets of H , then the FD $X \rightarrow Y$ is in $S+$ if and only if it holds in all relation universes over H in which all FD's in S hold.

If X is a finite set of attributes and S a finite set of FD's then the completion of X under S , written as $X+$, is the smallest superset of X such that if $Y \rightarrow Z$ in S and $Y \subseteq X+$ then $Z \subseteq X+$.

The completion of an attribute set can be used to compute if a certain dependency is in the closure of a set of FD's.

It can be proved that for a given header H and a set S of FD's that only contain subsets of H it holds that $X \rightarrow Y$ is in $S+$ if and only if $Y \subseteq X+$.

Given a header H and a set of FD's S that only contain subsets of H an irreducible cover of S is a set T of FD's such that:

1. $S+ = T+$
2. there is no proper subset U of T such that $S+ = U+$,
3. if $X \rightarrow Y$ in T then Y is a singleton set and
4. if $X \rightarrow Y$ in T and Z a proper subset of X then $Z \rightarrow Y$ is not in $S+$.

7.8.1 Full Functional Dependencies

A functional dependency $X \rightarrow Y$ is a *full functional dependency* if the removal of any attribute A from X implies that the functional dependency no longer holds. That is:

$$\text{if } X \rightarrow Y \quad \exists A \in X \ni (X - \{A\}) \not\rightarrow Y$$

7.8.2 Partial Functional Dependencies

A functional dependency $X \rightarrow Y$ is a *partial functional dependency* if some attribute A from X can be removed while the functional dependency still holds. That is:

$$\text{if } X \rightarrow Y \quad \exists A \in X \ni (X - \{A\}) \rightarrow Y$$

The identification of partial dependencies is used in the process of database normalisation outlined in section 7.17.

7.8.3 Transitive Functional Dependencies

A functional dependency $X \rightarrow Y$ in a relation is a *transitive dependency* if there is a set of attributes Z that is neither a candidate key nor a subset of any key of R while both $X \rightarrow Z$ and $Z \rightarrow Y$ hold. Attribute Y is transitively dependent on attribute X .

7.9 Database normalisation and Design

Database normalisation is usually performed when designing a relational database, to improve the logical consistency of the database design and the transactional performance.

7.10 The data model

A data model consists of tools and languages for describing:

1. Conceptual and external schemas
2. Constraints
3. Operations on data
4. An optional storage definition language which allows the database designer to interact with the physical data storage schema.

Refer to Lewis et al. [72, page 57].

7.11 Insertion, Deletion and Update Operations on Relations

The contents of the relations in a database typically vary over time and are created, changed and deleted via standard operations i.e. the insertion, deletion and update operations.

7.11.1 Inserting a Tuple into a Relation table

For a relation with schema $\mathbf{R} = \{A_1, A_2, A_3, \dots, A_n\}$ the format for the insert operation is:

$$\text{INSERT INTO relation name } (A_1 = v_1, A_2 = v_2, A_3 = v_3 \dots, A_n = v_n)$$

The values $v_1, v_2, v_3 \dots, v_n$ must belong to the domain of the attributes $A_1, A_2, A_3, \dots, A_n$. The operation adds a new tuple t to the relation with $t(A_i) = v_i$

This operation is not guaranteed to succeed because problems with values outside domains, incomplete tuples, duplicate primary keys, unique defined attribute columns and an undefined relation name, may arise.

7.11.2 Deleting a Tuple from a Relation table

The DELETE operation which removes a tuple from a relation has the format:

DELETE FROM relation name WHERE search condition

The search-condition can specify the values of the key attributes of other attributes of the tuple(s) which needs to be deleted from the relation.

This operation is also not guaranteed to succeed because problems with tuples not in a relation, foreign key referencing and an undefined relation name, may occur.

7.11.3 Updating a Tuple of a Relation Table

The update operation for a tuple in a relation changes a value of one or more of its attributes. The tuple which must be changed needs to be identified via one of its keys or attributes.

The format of this operation reads:

UPDATE relation name SET column name = new-value WHERE search condition

Updating can also be performed using a combination of the delete and insert operations and can fail due to the same reason that these operations can fail.

7.12 Specification of Attribute Domains

The domain of an attribute defines the characteristics of the values that a table column may contain. In database implementations the domain is typically implemented using a specified data type. Standardisation specifications by organisation such as American National Standards Institute (ANSI) and the International Standards Organisation (ISO) exist, but implementations differ and vary from vendor to vendor.

Some typical standard data types are listed in table 7.2.

Extensive documentation can be found in database program manuals such as e.g. The PostgreSQL Global Development Group [121].

7.13 Relational Algebra, Calculus and Relational Operations

Relational operations are a set of operations which are used to manipulate relations which are represented as tables in a database. Basic operations and set operations are defined. The basic operations include the selection, projection and equijoin operations treated here and then the set based relational operations discussed later. The relational operations form part of the relational algebra which is important for the logical understanding of the inner workings of a Relational Database Management System (RDBMS) and the Structured Query Language (SQL) used to define data input and queries for these databases.

The operands of the operations are relations. The operations include unary operations such as:

- selection
- projection

and binary (set) operations such as:

- union
- set difference
- Cartesian product
- intersection

These operations have no effect on the originating relation(s). Five of these operations are fundamental i.e. selection, projection, Cartesian product, union and set difference. All other operations can be defined in terms of these operations.

Table 7.3 from Elmasri and Navathe [35], summarises the development in this section.

bit	fixed-length bit string
bit varying	variable-length bit string
boolean	logical Boolean (true/false)
char	single character
character varying (n)	variable-length character string up to n characters
character(n)	fixed-length character string of n characters
varchar(n)	character string with varying number of characters up to n characters
date	date value
double precision	double precision floating-point number value as implemented by system
integer	signed four-byte fixed point integer numerical value as implemented by system
interval(p)	time span
numeric [(p, s)]	exact numeric with selectable precision
decimal [(p, s)]	see numeric
real	single precision floating-point numerical value as implemented by system
smallint	signed two-byte fixed point integer numerical value as implemented by system
time [(p)] [with or with out time zone]	time of day optionally including time zone
timestamp [(p)] [with time zone]	date and time, including time zone

Table 7.2: Some standard SQL data types

7.13.1 The Selection Operation

Mathematically the unary selection operation $\sigma_{A=a}(\mathbf{R})$ i.e. the selection of \mathbf{R} on A is defined as:

$$\sigma_{A=a}(\mathbf{R}) = \{t \in \mathbf{R} \mid t(A) = a\} \quad (7.13.1)$$

The selection operation produces a new relation (table) where the rows (tuples) are a subset of the rows of the original relation which have a particular specified value for an attribute.

The schema of the new relation $\sigma_{A=a}(\mathbf{R})$ is the same as that of the old relation \mathbf{R} and it has the same attributes.

7.13.2 The Projection Operation

The projection operation $\pi_X(\mathbf{R})$ of a relation \mathbf{R} onto a set \mathbf{X} of its attributes is defined as:

$$\begin{aligned} &\text{given: } \mathbf{X} = \{A_1, A_2, A_3, \dots, A_k\} \subset \mathbf{A}; k < n \\ \pi_X(\mathbf{R}) &= \{t(X) \mid t \in \mathbf{R}\} \end{aligned} \quad (7.13.2)$$

The entries for tuple j of $\pi_X(\mathbf{R})$ are formed by selecting the entries $t_j(A_1), t_j(A_2), t_j(A_3), \dots, t_j(A_k)$.

The projection operation produces a new relation (table) where the columns (attributes) are a subset of the columns of the original relation.

7.13.3 Tuple Concatenation Operation

Given two tuples $s = (s_1, s_2, s_3, \dots, s_n)$ and $t = (t_1, t_2, t_3, \dots, t_m)$ the concatenation of s and t is the $(m+n)$ tuple defined as:

$$st = (s_1, s_2, s_3, \dots, s_n, t_1, t_2, t_3, \dots, t_m) \quad (7.13.3)$$

7.14 Set Operations on Relations

The set based database relational operations are constituted in parallel to the relational algebra operations defined in chapter 4, section 4.17.4. The operations involves two relations (sets), which can be nested and the result of each set operation is a new relation.

7.14.1 Set Union Operation

Given two relations R and S with union compatible schemas (the degrees of the relations must match), the union of these two relations is denoted by $R \cup S$ and is the set of tuples that are present in R or S or in both relations.

$$R \cup S = \{t \mid (t \in R) \vee (t \in S)\} \quad (7.14.1)$$

The union of relations is strictly not a commutative operation. The schema of the union is the set of attributes of the schema of relation R , which implies that the schema of $R \cup S$ can differ from that of $S \cup R$. ($R \cup S \neq S \cup R$). As required for a relation no duplicate entries are found in the union of two relations.

If R and S have i_R and i_S tuples, respectively, the union will have a maximum of $i_R + i_S$ tuples.

7.14.2 Set Difference Operation

Given two relations R and S with union compatible schemas, the difference of these two relations is denoted by $R - S$ and is the set of tuples that are present R but not in S .

$$R - S = \{t \mid (t \in R) \wedge (t \notin S)\} \quad (7.14.2)$$

The order in which the relations for the difference are named is important because the operation is not commutative and $R - S \neq S - R$.

If R and S have i_R and i_S tuples, respectively, the difference will have a maximum of $i_R - i_S$ tuples.

7.14.3 Set Intersection Operation

Given two relations R and S with union compatible schemas, the intersection of these two relations is denoted by $R \cap S$ and is the set of tuples that are present in both relations.

$$R \cap S = \{t \mid (t \in R) \wedge (t \in S)\} \quad (7.14.3)$$

The intersection of two relations can produce an empty relation.

The intersection of relations is a commutative operation because $R \cap S = S \cap R$.

The intersection can be derived from the set difference $S - (R - S)$.

7.14.4 Set Cartesian Product Formation Operation

The *Cartesian product* of two non-empty relations R and S is denoted by $R \otimes S$ and is defined as the relation formed by concatenating the every tuple of relation R with every tuple of relation S .

In mathematical form this reads:

$$R \otimes S = \{pq \mid (p \in R) \vee (q \in S)\} \quad (7.14.4)$$

The logical description of the relation produced by a Cartesian product operation may not necessarily make sense. The operation is sometimes useful when followed by a selection operation that matches values of attributes coming from the component relations.

The degree of the Cartesian product of two relations is the sum of the degrees of the two original relations (the operands) and the cardinality is the product of the cardinality of the original relations.

If R has i_R tuples and m_R attributes and S has i_S tuples and m_S attributes, the resulting relation has $i_R * i_S$ tuples and $m_R + m_S$ attributes.

7.14.5 The Join Operation

The join operation which is denoted by \bowtie combines attributes of two relations into one. Tuples in R are combined with related tuples in S .

It is different from a Cartesian Product ($R \times S$) as it involves a selection predicate i.e. a Select on a Cartesian Product.

It is useful for relational algebra, but memory-intensive in the practical world, so vendors try to optimise for these using 'query optimisation'.

Given two relations $R(A_1, A_2 \dots A_n)$ and $S(B_1, B_2 \dots B_m)$ the general form of a join operation is:

$$R \bowtie_{\langle \text{join condition} \rangle} S$$

The result of a join operation is a relation Q with $n + m$ attributes $Q(A_1, A_2 \dots A_n, B_1, B_2 \dots B_m)$ in the given order. Q has one tuple for each combination of tuples, one from R and one from S , whenever the join condition is satisfied by the combination of tuples. The *join condition* is specified on attributes from both relations R and S and is evaluated for each combination of tuples. When the join condition evaluates as true the combined tuple is included in the resulting relation Q .

The Theta Join Operation

The *theta join* condition is of the form:

$$\langle \text{condition} \rangle \text{ AND } \langle \text{condition} \rangle \text{ AND } \dots \text{ AND } \langle \text{condition} \rangle$$

Each condition is of the form $A_i \theta B_j$ where A_i is an attribute of R and B_j an attribute of S having the same domain. θ is taken from the set of comparison operators. $\theta \in \{=, <, \leq, >, \geq, \neq\}$

The resulting relation from the theta join operations contains the sum of the degrees of the two operand relations.

The Equijoin Operation

The equijoin operation is a theta join operation with equality used as the comparison operator.

The equijoin operation combines two relations on all their common attributes. The join consists of all the tuples formed when the tuples of the first relation are concatenated to those of the second relation for a given common set of attributes X . Common attributes are defined as attributes which have the same domain and meaning.

Given a relation $r(R)$ with a set of attributes R and a relation s with a set of attributes S and a set X of the common attributes of R and S i.e. $X = R \cap S$. For every tuple of the relation r *Joins* one must have:

- $\exists t_r$ of r *Joins* so that $t(R) = t_r$
- $\exists t_s$ of r *Joins* so that $t(S) = t_s$
- $t_r(X) = t_s(X)$

For relations R and S the join operation is defined mathematically as:

$$R \text{Join} S = \{rs \mid s \in R \text{ and } r(R \cap S) = s(R \cap S)\} \quad (7.14.5)$$

The tuple concatenation operator \sqcup defined in section 7.13.3 is used in equation 7.14.5. The resulting relation from the equijoin operations contains the sum of the degrees of the two operand relations.

The Natural Join Operation

The natural join is denoted by $*$ and can be defined as:

$$Q \leftarrow R *_{(\langle \text{list1} \rangle)(\langle \text{list2} \rangle)} S$$

For a *natural join* the relations being joined need to have one attribute (domain) name in common and the common attribute is the one being compared to see if a new tuple will be inserted in the resulting relation. It might be necessary to rename an attribute to have a common attribute name. The resulting relation from the natural join contains the sum of the degrees of the two relations minus the duplicate attributes after the first attribute.

The Outer Join Operation

The *outer join* was developed to form the union of tuples from two relations if the relations are not union compatible i.e. the relations are partially compatible. The list of compatible attributes needs to include a key for each relation of the join. Tuples from the operand relations of the join with the same key appear only once in the join result, while attributes which are not union compatible are kept in the result but tuples that have no values for attributes are filled with null entries.

Three types of outer joins i.e. left outer join, right outer join and full outer join are defined.

The left outer join operation keeps every tuple in the first or left relation R in $R \bowtie S$. If no matching tuple is found in S then the attributes of S in the join result is filled with null entries. The left outer join is denoted by the $\sqsupset\bowtie$ symbol.

The right outer join operation keeps every tuple in the second or right relation S in $R \bowtie S$. If no matching tuple is found in R then the attributes of R in the join result is filled with null entries. The right outer join is denoted by the $\bowtie\sqsubset$ symbol.

The full outer join keeps every tuple in both relations R and S where no matching tuples are found filling in null entries as needed. The full outer join is denoted by the $\sqsupset\bowtie\sqsubset$ symbol.

The Semijoin Operation

The semijoin operation was added to help manage distributed database transactions deployed over networked environments. If local relation R needs to be joined to remote relation S , the approach is to send the joining column of relation R only to the site where relation S is located. Only the transferred column is then joined to S at the remote site. The join attributes and the attributes of S required in the result are projected out of the relation S and then transferred back to the site hosting R . This approach can minimise data transfer when a small fraction of the tuples of S are involved in the join.

The semijoin is denoted by the \triangleright symbol. The semijoin shown below is equivalent to the database relational algebra expression shown.

$$R \triangleright_{A=B} S \sim \pi_R(R \bowtie_{A=B} S)$$

Join Operation Example

Given the relations R , S and T with $R(a, b)$, $S(b, c)$, $T(c, d)$ with attributes a, b, c, d as indicated.

Using e.g. the notation $R.a$ to indicate attribute a of relation R the join operations yield:

- Natural Join: $M(a, R.b, c) \leftarrow R \bowtie S$
- Theta Equijoin: $N(b, S.c, T.c, d) \leftarrow S \bowtie_{S.c=T.c} T$
- Theta Less than Join : $O(b, S.c, T.c, d) \leftarrow S \bowtie_{S.c < T.c} T$
- Left Outer Join: $N(b, S.c, T.c, d) \leftarrow S \sqsupset\bowtie_{S.c=T.c} T$
- Right Outer Join: $N(b, S.c, T.c, d) \leftarrow S \bowtie\sqsubset_{S.c=T.c} T$
- Full Outer Join: $N(b, S.c, T.c, d) \leftarrow S \sqsupset\bowtie\sqsubset_{S.c=T.c} T$
- Semi Join: $P(b, S.c) \leftarrow S \triangleright_{S.c < T.c} T$

Note that expression simplification is an important strategy used in relational algebra operation implementations. Database query optimisation techniques are used and can affect the running time of queries by an order of magnitude or more. Early ‘selection’ reduces the number of tuples while early ‘projection’ reduces the number of domains to be processed.

The Division Operation

The *division operation* is applied to two relations $R(Z) \div S(X)$ where for the attribute sets X and Y $X \subseteq Y$ holds. Taking $Y = Z - X$ i.e. $Z = X \cup Y$ the result of the division is a relation $T(Y)$ which includes a tuple t_R if it appears in R and with $t_R(Y) = t$ and with $t_r(X) = t_s$ for every tuple t_s in S . This implies that for a tuple t to appear in the result of a division T , the values of t must appear in R in combination with every tuple in S .

An example where the division operation is used, is the query ‘Retrieve the names of the employees who work on all projects which employee E works on’.

<i>Operation</i>	<i>Purpose</i>	<i>Notation</i>
SELECT	Selects all tuples that satisfy the selection condition from a relation R .	$\sigma_{\langle \text{selection condition} \rangle} (R)$
PROJECT	Produces a new relation with selected attributes of relation R and removes all duplicate tuples.	$\pi_{\langle \text{attribute list} \rangle} (R)$
THETA JOIN	Produces all tuples from relation R_1 and R_2 which satisfy the join condition.	$R_1 \bowtie_{\langle \text{join condition} \rangle} R_2$
EQUIJOIN	Produces all tuples from relation R_1 and R_2 which satisfy the join condition with only equality comparisons.	$R_1 \bowtie_{\langle \text{join condition} \rangle} R_2$ or $R_1 \bowtie_{(\langle \text{join attributes1} \rangle, \langle \text{join attributes2} \rangle)} R_2$
NATURAL JOIN	Same as the EQUIJOIN except that the attributes of R_2 are not included in the resulting join. If join attributes have the same name they need not be specified at all.	$R_1 *_{\langle \text{join condition} \rangle} R_2$ or $R_1 *_{(\langle \text{join attributes1} \rangle, \langle \text{join attributes2} \rangle)} R_2$ newline or $R_1 * R_2$
UNION	Produces a relation that includes all tuples in R_1 and R_2 or both R_1 and R_2 . R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all tuples both in R_1 and R_2 . R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all tuples in R_1 that are not in R_2 . R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 . All possible combinations of tuples from R_1 and R_2 are included.	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t(X)$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$ where $Z = X \cup Y$	$R_1(Z) \div R_2(Y)$

Table 7.3: Operations of Database Relational Algebra

The division operation can be expressed as a sequence of the basic operations π , \times and $-$ as follows:

$$\begin{aligned}
 T_1 &\leftarrow \pi_Y (R) \\
 T_2 &\leftarrow \pi_Y ((S \times T_1) - R) \\
 T &\leftarrow T_1 - T_2
 \end{aligned}$$

7.14.6 Relation and Attribute Rename Operation

A *rename* operation can be defined to rename relations and attributes of relations. This is useful when intermediate results of relational algebra operations are processed and identifiers need to match up.

The rename operation is denoted by ρ . The format of a rename applied to a relation of degree n is shown below.

$$\begin{aligned}
 &\rho_{S(B_1, B_2, B_3, \dots, B_n)} (R) \\
 &\text{or } \rho_S(R) \\
 &\text{or } \rho_{(B_1, B_2, B_3, \dots, B_n)} (R)
 \end{aligned}$$

7.14.7 Grouping and Aggregation Operations

Database reports typically require grouping of report lines and sub-totalling and totalling of selected columns of report lines.

The specification of mathematical aggregate functions on collections of values from a relation or database, does not form part of the relational algebra. Common functions applied to collections of numeric values include SUM, AVERAGE, MAXIMUM and MINIMUM. The COUNT function is used to count sets of tuples or values.

The grouping of tuples on a given criterion to which aggregate functions are applied is also required. The format of this operation is given below. Grouping is denoted by \mathcal{F} .

$$\langle \text{grouping attributes} \rangle \mathcal{F} \langle \text{function list} \rangle (R)$$

The processing of null tuple values as operands in mathematical operations needs special care and must be specified to make logical sense and prevent erroneous processing or system error messages.

7.15 Relational Calculus

Relational calculus is a high-level, declarative and non-procedural formal database query language. It developed from a branch of symbolic logic known as predicate calculus. A predicate is a truth-valued function with arguments. Predicate arguments are replaced with values to obtain a proposition. A proposition is either true or false.

Relational calculus is identical in its expressive power with relational algebra. It declares what is to be retrieved, not how to retrieve it and requires a well-formed formula.

Relational calculus forms the basis for *relationally complete languages* such as IBM Query-ByExample. A language is relationally complete if any query expressed by the relational calculus can also be expressed by the language.

Relational calculus formulations of operations defined in relational algebra have been formulated. Two typical formats are in use i.e. the Tuple Relational Calculus (TRC) developed by Codd and the Domain Relational Calculus (DRC).

DRC has become the basis of visual query languages such as IBM Query-ByExample and is also used in products such as Microsoft Access (Microsoft Corporation [82]) and Corel WordPerfect Office X3 - Professional Edition Paradox (Corel Corporation [25])(previously known as Borland Paradox).

DRC is similar to TRC but uses domain variables.

7.15.1 Tuple Relational Calculus

The general format of an expression in *tuple relational calculus* is of the form:

$$\{T \mid F(T)\} \text{ or } \{t_1.A_1, t_2.A_2, \dots, t_n.A_n, \mid COND(t_1, t_2, \dots, t_n, t_{n+1}, \dots, t_{n+m})\}$$

This is interpreted as: Find the set of all tuples T such that the condition or formula F or $COND$ is true.

A formula is made up of predicate calculus atoms which can be:

1. An atom of the form $R(t_i)$ where R is a relation name and t_i is a tuple variable.
2. An atom of the form $t_i.A$ **operand** $t_j.B$, where **operand** $\in \{=, >, \geq, <, \leq, \neq\}$ and t_i and t_j are tuple variables. A is a relation on which t_i ranges and B is a relation on which t_j ranges.
3. An atom of the form $t_i.A$ **operand** c , or c **operand** $t_j.B$ where **operand** $\in \{=, >, \geq, <, \leq, \neq\}$. A is a relation on which t_i ranges and B is a relation on which t_j ranges and c is a constant value.

Atoms are connected via the logical operators AND (\wedge), OR (\vee) and NOT (\sim) or (\neg) and the universal qualifier (\forall - for all) and existential (\exists - there exists) of predicate logic is also used. The symbols \bigvee_x and \bigwedge_x are alternatives for the existential qualifier and universal qualifier respectively.

1. If F is a formula then $(\exists t)(F)$ is also a formula. It evaluates to *true*, if for *some* tuple t assigned to a free occurrence of t in F , F evaluates to true. Otherwise $(\exists t)(F)$ evaluates to false.
2. If F is a formula then $(\forall t)(F)$ is also a formula. It evaluates to *true*, if for *every* tuple t assigned to a free occurrence of t in F , F evaluates to true. Otherwise $(\forall t)(F)$ evaluates to false.

The standard transformations of the universal and existential qualifiers to form equivalent expressions apply to the tuple relational calculus.

$$\begin{aligned}
(\forall x) (P(x)) &\equiv \neg (\exists x) (\neg P(x)) \\
(\exists x) (P(x)) &\equiv \neg (\forall x) (\neg P(x)) \\
(\forall x) (P(x) \wedge Q(x)) &\equiv \neg (\exists x) (\neg P(x) \vee \neg Q(x)) \\
(\forall x) (P(x) \vee Q(x)) &\equiv \neg (\forall x) (\neg P(x) \wedge \neg Q(x)) \\
(\exists x) (P(x) \vee Q(x)) &\equiv \neg (\exists x) (\neg P(x) \wedge \neg Q(x)) \\
(\exists x) (P(x) \wedge Q(x)) &\equiv \neg (\forall x) (\neg P(x) \vee \neg Q(x)) \\
&\text{and} \\
(\forall x) (P(x)) &\Rightarrow (\exists x) (P(x)) \\
\neg (\exists x) (P(x)) &\Rightarrow \neg (\forall x) (P(x))
\end{aligned} \tag{7.15.1}$$

Tuple relational calculus examples

Two basic examples using the tuple relational calculus formulation are listed below.

Single relation: List the names of all managers who earn more than 25,000 -
 $\{S.Name \mid Staff(S) \wedge S.position = 'manager' \wedge Salary > 25000\}$

Multiple relations: List names of staff who manage properties for rent in Glasgow -
 $\{S.Name \mid Staff(S) \wedge \exists (P)(PropertyForRent(P) \wedge (P.staffno = S.staffno) \wedge P.city = 'Glasgow')\}$

7.15.2 Domain Relational Calculus

For the *domain relational calculus* or domain calculus variables range over single values from domains of attributes.

The general format of an expression in *domain relational calculus* is:

$$\{d_1, d_2, \dots, d_n \mid F(d_1, d_2, \dots, d_n)\} \text{ or } \{d_1, d_2, \dots, d_n \mid COND(d_1, d_2, \dots, d_n, d_{n+1}, \dots, d_{n+m})\}$$

This is interpreted as: Find the domain variables d_1, d_2, \dots, d_n such that the condition or formula F or $COND$ is true. The predicate requires finding a tuple containing a value in each domain that satisfies the proposition.

A formula is made up of predicate calculus atoms which can be:

1. An atom of the form $R(d_1, d_2, \dots, d_n)$ where R is a relation name of degree j for each $d_i, 1 \leq i \leq j$ is a domain variable.
2. An atom of the form d_i **operand** d_j , where **operand** $\in \{=, >, \geq, <, \leq, \neq\}$ and d_i and d_j are domain variables.
3. An atom of the form d_i **operand** c , or c **operand** d_j where **operand** $\in \{=, >, \geq, <, \leq, \neq\}$. d_i and d_j are domain variables and c is a constant value.

To make domain relational calculus more precise the comma separators in the lists of variables are dropped to write:

$$\{d_1, d_2, \dots, x_n \mid R(x_1 x_2 x_3) \text{ and } \dots\}$$

As for the tuple relational calculus atoms evaluate to truth values and qualifiers are used as outlined in section 7.15.1.

Domain relational calculus examples

Two basic examples using the domain relational calculus formulation are listed below.

Find the names of managers who earn more than 25000 -
 $\{N \mid (\exists N, pos, sal)(Staff(N, pos, sal) \wedge pos = 'manager' \wedge sal > 25000)\}$

List all cities where there is either a branch office or a property for rent:
 $\{city \mid (Branch(bN, st, city, pc) \vee (PropertyForRent(pN, st1, city, pc1, rms)))\}$

Operation	Algebra	Tuple Relational Calculus (TRC)	Domain Relational Calculus (DRC)
<i>Selection</i>	$\sigma_{Condition}(R)$	$\{T \mid R(T) \wedge Condition_1\}$	$\{X_1, \dots, X_n \mid R(X_1, \dots, X_n) \wedge Condition_2\}$
<i>Projection</i>	$\pi_{A,B,C}(R)$	$\{T.A, T.B, T.C \mid R(T)\}$ Assume R has five attributes with A, B, C the first three	$\{X, Y, Z \mid \exists V \exists W (R(X, Y, Z, V, W))\}$
<i>Cartesian product</i>	$R \times S$	$\{T.A, T.B, T.C, V.D, V.E \mid R(T) \vee S(V)\}$ Assume R has attributes A, B, C and S attributes D, E	$\{X, Y, Z, V, W \mid R(X, Y, Z) \wedge S(V, W)\}$
<i>Union</i>	$R \cup S$	$\{T \mid R(T) \vee S(T)\}$	$\{X_1, \dots, X_n \mid R(X_1, \dots, X_n) \vee S(X_1, \dots, X_n)\}$
<i>Set difference</i>	$R - S$	$\{T \mid R(T) \vee (\neg S(T))\}$	$\{X_1, \dots, X_n \mid R(X_1, \dots, X_n) \vee \neg S(X_1, \dots, X_n)\}$
<i>Division</i>	R/S where R has attributes A, B and S only has attributes B	$\{T.A \mid R(T) \wedge \forall X \in S \exists Y \in R(Y.B = X.B \wedge Y.A = T.A)\}$	$\{X, Y, Z \mid R(X_1, \dots, X_n) \wedge S(X_1, \dots, X_n) \exists Y \in R(Y_l, Y_m, Y_n) = R(X_l, X_m, X_n) \wedge (X, Y, Z) = (X_l, X_m, X_n)\}$

Table 7.4: Comparison of Relational Algebra and Calculi

7.15.3 Relational Algebra and Relational Calculus

The relational algebra and relational calculus have equivalent outcomes for selected specifications.

Table 7.4 outlines the formats of the three approaches for equivalent operations.

7.16 Structured Query Language (SQL)

The operations defined in sections 7.11 to 7.14.2 are defined in the Structured Query Language (SQL) which is used to describe and manipulate data sets in database systems. The details of the syntax, semantics and other implementation aspects of the SQL language as used in chapter 13 and appendix J, is not discussed here, but can be found in references such as Houlette [56], Forta [40] and Plew and Stephens [95].

7.17 The database normalisation process

To eliminate data redundancy and potential data update anomalies several *normal forms* for database schemas are defined in database theory. If a schema is in one of the normal forms it has certain predictable properties. Codd [24] proposed three normal forms where each normal form eliminates more anomalies than the previous one.

Normalisation procedures provide database designers with a formal framework for analysing relational schemas based on their keys and functional dependencies among their attributes and a series of tests which can be carried out on schemas so that a relational database can be normalised to any selected degree.

The *first normal form (1NF)*, introduced by Codd, is equivalent to the definition of the relational data model.

The *second normal form (2NF)* was an attempt to eliminate some potential anomalies. This normal form is of no practical use.

The *third normal form (3NF)* was originally thought to be the ultimate normal form but Boyce and Codd realised that it still contained undesirable combinations of functional independencies and this led to the introduction of the *Boyce-Codd normal form (BCNF)*. Although BCNF is more desirable it is not always achievable without paying a price somewhere else. Algorithms are available to convert schemas which harbour various undesirable properties into 3NF and BCNF. There are however trade-offs associated with the conversion process.

The *fourth normal form (4NF)* deals with the problem of other dependencies which are not functional dependencies and extends BCNF.

The process of normalisation should also confirm other properties which relational schemas should possess. Two of these properties are:

- The *lossless join* property ensures that the problem of spurious tuple generation does not exist. Spurious tuples contain incorrect information when a natural join of a poorly decomposed relation is made.
- The *dependency preservation property* which ensures that each functional dependency is still represented in relations after decomposition

7.17.1 The First Normal Form

A relation $r(R)$ is said to be in *first normal form (1NF)* if and only if every entry in the relation (the intersection of a tuple and an attribute column) has at most a single value i.e. all its attributes are based on a simple domain and the domain can only include atomic (simple, indivisible) values. If all relations of a database are in 1NF the database is said to be in 1NF. To normalise a table all repeating groups of attributes need to be removed. Typically the table needs to be flattened (all repeating groups are removed by filling in the missing entries) and a suitable primary composite key needs to be defined for the table to make it a relational table again.

As an alternative the table can be subdivided (decomposed) into two tables replacing the original table. The one table contains the table identifier of the original table and all the *non-repeating* attributes. The other table contains a copy of the table identifier and all the *repeating attributes*. The second approach can be preferred due to efficiency with less data redundancy.

Refer to the example below taken from Mata-Toledo and Cushman [80]. Table 7.5 is decomposed into tables 7.6 and 7.7.

Proj-ID	Proj-Name	Proj-mgr-ID	Emp-ID	Emp-Name	Emp-Dept	Emp-HrlyRate	Total-Hrs
100	E-commerce	789487453	123423479	Heydary	MIS	65	10
			980808980	Jones	TechSupport	45	6
			123423479	Alexander	TechSupport	35	6
			123423479	Johnson	TechDoc	30	12
110	Distance-Ed	820972445	432329700	Mantle	MIS	50	5
			689231199	Richardson	TechSupport	35	12
			712093093	Alexander	TechDoc	30	8
120	Cyber	980212343	834920043	Lopez	Engineering	80	4
			380802233	Harrison	TechSupport	35	11
			553208932	Oliver	TechDoc	30	12
			123423479	Heydary	MIS	65	07
130	Nitts	550227043	340783453	Shaw	MIS	65	7

Table 7.5: The PROJECT Table with duplicate entries

Proj-ID	Proj-Name	Proj-mgr-ID
100	E-commerce	789487453
110	Distance-Ed	820972445
120	Cyber	980212343
130	Nitts	550227043

Table 7.6: The PROJECT Table

Proj-ID	Emp-ID	Emp-Name	Emp-Dept	Emp-HrlyRate	Total-Hrs
100	123423479	Heydary	MIS	65	10
100	980808980	Jones	TechSupport	45	6
100	123423479	Alexander	TechSupport	35	6
100	123423479	Johnson	TechDoc	30	12
110	432329700	Mantle	MIS	50	5
110	689231199	Richardson	TechSupport	35	12
110	712093093	Alexander	TechDoc	30	8
120	834920043	Lopez	Engineering	80	4
120	380802233	Harrison	TechSupport	35	11
120	553208932	Oliver	TechDoc	30	12
120	123423479	Heydary	MIS	65	07
130	340783453	Shaw	MIS	65	7

Table 7.7: The PROJECT-EMPLOYEE Table

Data anomalies in 1NF relations

Data anomalies refer to undesirable effects on data due to some relational operations. Insertion/deletion and update anomalies are identified.

Insertion deletion anomalies occur where functional dependencies occur in data tuples and e.g. the employee and his/her department needs to be inserted in a project record. It is also difficult to insert a department which has no employees yet.

A deletion anomaly occurs when the (employee, department) tuple is deleted for the last employee of a department and the information on the department is lost from the database.

Update anomalies occur when functional dependencies between attributes imply updates for a number of tuples when the data is carried is duplicated. E.g. the employee department needs to be changed in each project record if the employee and his/her department is contained in each project record.

7.17.2 The Second Normal Form

A relation $r(R)$ is in *Second Normal Form* (2NF) if and only if:

1. $r(R)$ is in 1NF
2. No nonprime attribute of the relation is partially dependent on any key, i.e. each nonprime attribute in R is fully functional dependent upon every key (including candidate keys).

Refer to section 7.8 for a definition of functional dependency of attributes.

The test for 2NF involves finding functional dependencies whose left-hand side attributes are part of the primary key of the relation. For keys consisting of single attributes this test is not required.

If a relation is not in 2NF it can be 2NF normalised into a number of 2NF relations where the non-prime attributes are associated only with part of the primary key on which they are fully functional dependent.

7.17.2.1 Data anomalies in 2NF relations

Relations in 2NF are still subject to data anomalies. Insertion/deletion and update anomalies can still be identified for the applicable relational operations as for the 1NF form.

7.17.3 The Third Normal Form

A relation $r(R)$ is in *Third Normal Form* (3NF) if and only if:

1. $r(R)$ is in 2NF
2. No nonprime attribute is transitively dependent on the key of the relation.

A relation schema R is in third normal form (3NF) if whenever a nontrivial functional dependency $X \rightarrow A$ holds in R , then

1. X is a superkey of R or
2. A is a prime attribute of R , i.e. it is part of any key of R

A relation schema R violates 3NF if a functional dependency $X \rightarrow A$ holds in R and violates the two requirements above as shown below.

1. X is not a superset of a key of R
2. A is a nonprime attribute of R

An alternative definition of 3NF can be stated. A relation schema R is in 3NF if every nonprime attribute of R is fully functionally dependent on every key of R and it is nontransitively dependent on every key of R .

Data anomalies in 3NF relations

Relations in 3NF are still susceptible to data anomalies when the relations have two overlapping candidate keys or when a nonprime attribute functionally determines a prime attribute.

7.17.4 The Boyce-Codd Normal Form

The Boyce-Codd Normal Form (BCNF) was proposed as a simpler form of 3NF to eliminate the stated anomalies for 3NF.

A relation schema R with associated relation $r(R)$ is in Boyce-Codd Normal Form (BCNF) if and only if the following conditions hold:

1. The relation is 1NF
2. For every functional dependency of the form $X \leftarrow A$ either X is a superkey of r or $A \subset X$

The following holds for a relation in BCNF:

- All nonprime attributes are fully dependent on every key
- All prime attributes are fully dependent on all keys of which they are not part

The set of 3NF relations form a proper subset of the BCNF relations. All BCNF relations are 3NF but not all 3NF relations are BCNF which is more restrictive than 3NF.

7.17.5 The Fourth Normal Form

The fourth normal form will not be discussed here. The reader is referred to Elmasri and Navathe [35] and Lewis et al. [72] for details of this normal form.

7.18 Database Transaction Processing

The theory, modelling and technology implementation of database transactions form an important part of typical business system database functionality. It is not discussed here as it falls outside the scope of this study.

7.19 Additional reference material

This chapter only dealt with the basic background on database theory to support the material developed in chapter 13.

Further material can be found in references such as:

- Lewis et al. [72]
- Date [27]
- Mata-Toledo and Cushman [80]

- Paredaens et al. [93]
- Pratt [97]
- Rennhackkamp [101]
- Green [49]
- Dittrich et al. [30]
- Kim and Lochovsky [68]
- Elmasri and Navathe [35]
- Stanczyk [115]
- Maier [79]
- Patrick [94]
- Abiteboul et al. [1]
- Helman [54]
- Atzeni and De Antonellis [9]
- Dutka and Hanson [34]

Part II

Management models and techniques -
development technology demonstration

Chapter 8

Overview of Part II

In this part the application of the basic theory set out in part I is outlined.

Basic tools which apply the theory of part I are developed to demonstrate typical approaches to management related models relevant to consulting engineering service enterprises.

The following tools and techniques for management models are developed in this part.

- Tools for relational algebra applications, using boolean matrix representation of relations, programmed in MATLAB.
- String literal processing functionality programmed in MATLAB.
- An Engineering process model for computation of task schedules and related data, using the relational algebra and related graph theory approach, programmed in MATLAB
- Application of trees, defined in graph theory terms, to represent management hierarchies and define structures for management reporting functionality.
- A client-server or client-only database development of the engineering process model to facilitate reporting using commercially available database software such as Microsoft Access and spreadsheet software such as Microsoft Excel.

Extended details of e.g. MATLAB code developed are contained in the appendices.

Conclusion, recommendations and suggestions for further work

It will be shown in the addendum in part IV, that the basic technology demonstrated in this part is suitable as the basis of basic enterprise systems as well as management reporting and decision support systems for engineering services enterprises.

A full implementation of an enterprise management system falls outside the scope of this dissertation, but can be based on the concepts developed here.

The development of a user interface for the management reporting structures, generating SQL code linking to a database could also be attempted in future.

It should be possible to apply the path algebra theory set out in 5.9 to develop the reporting structures set up here using graph adjacency matrix manipulation.

The PLEP Engineering process model program (Eygelaaar [38]) can be extended to link to the database environment for reporting functionality.

Chapter 9

Relational Algebra MATLAB Tools

9.1 MATLAB boolean matrix relational algebra package (toolbox)

MATLAB does not support the computation of products of matrices with boolean entries.

For the purpose of demonstrating Relational Algebra operations in boolean matrix format the basic operations listed were programmed as MATLAB inline functions. This definition file can be accessed from every MATLAB program (.m) file to make the operations available in MATLAB function form. Alternatively .m files can set up in a library with each of the operations programmed separately.

The functions programmed are listed in table 9.2.

The implementation uses the normal MATLAB logical data representation and default logical data type operations. Refer to table 9.1. This is a purist approach where all entities dealt with are of datatype logical. MATLAB does not provide MOD1 overloads of its matrix multiplication and some other operations which are required for boolean matrix operations.

It will be possible to invoke the MATLAB sparse matrix functionality if required for large problems.

The boolean variable values, i.e. the terms true and one (1) and false and zero (0) are used on an interchangeable basis below.

The available basic boolean matrix operations are listed in table 9.3

9.2 MATLAB Relational Algebra Tools Code

Refer to Appendix C for the MATLAB code for the relational algebra functionality. An example of the use of the tools is also shown.

Table 9.1: MATLAB standard logical operations and functions on boolean (logical) variables used

<i>Routine</i>	<i>Usage</i>	<i>Parameters</i>
<i>not(A)</i> or $\sim A$	\sim Logical NOT. $\sim A$ is a matrix whose elements are 1's where A has zero elements, and 0's where A has non-zero elements. $B = \text{NOT}(A)$ is called for the syntax ' $\sim A$ ' when A is an object	A: logical variable or matrix
<i>and(A, B)</i> or $\&$	$\&$ Logical AND. $A \& B$ is a matrix whose elements are 1's where both A and B have non-zero elements, and 0's where either has a zero element. A and B must have the same dimensions unless one is a scalar. $C = \text{AND}(A,B)$ is called for the syntax ' $A \& B$ ' when A or B is an object	A, B: logical variables or matrices
<i>or(A,B)</i> or \parallel	Logical OR. $A \parallel B$ is a matrix whose elements are 1's where either A or B has a non-zero element, and 0's where both have zero elements. A and B must have the same dimensions unless one is a scalar. $C = \text{OR}(A,B)$ is called for the syntax ' $A \parallel B$ ' when A or B is an object	A, B: logical variables or matrices
<i>xor(S,T)</i>	XOR Logical EXCLUSIVE OR. $\text{XOR}(S,T)$ is the logical symmetric difference of elements S and T. The result is one where either S or T, but not both, is nonzero. The result is zero where S and T are both zero or nonzero. S and T must have the same dimensions (or one can be a scalar)	S, T: logical variables or matrices
<i>any(V)</i>	ANY True if any element of a vector is a nonzero number or is logical TRUE (1). ANY ignores entries that are NaN (Not a Number). For vectors, $\text{ANY}(V)$ returns logical TRUE (1) if any of the elements of the vector is a nonzero number or is logical TRUE. Otherwise it returns logical FALSE (0). For matrices, $\text{ANY}(X)$ operates on the columns of X, returning a row vector of 1's and 0's. For multi-dimensional arrays, $\text{ANY}(X)$ operates on the first non-singleton dimension. $\text{ANY}(X,\text{DIM})$ works down the dimension DIM. For example, $\text{ANY}(X,1)$ works down the first dimension (the rows) of X	A: logical variable or matrix
<i>all(A)</i>	ALL True if all elements of a vector are nonzero. For vectors, $\text{ALL}(V)$ returns 1 if none of the elements of the vector are zero. Otherwise it returns 0. For matrices, $\text{ALL}(X)$ operates on the columns of X, returning a row vector of 1's and 0's. For N-D arrays, $\text{ALL}(X)$ operates on the first non-singleton dimension. $\text{ALL}(X,\text{DIM})$ works down the dimension DIM. For example, $\text{ALL}(X,1)$ works down the first dimension (the rows) of X.	A: logical variable or matrix

Table 9.2: MATLAB - Basic Relational Algebra Operations using Boolean matrix representation of relations

<i>Routine</i>	<i>Usage</i>	<i>Parameters</i>
<i>productR(A, B)</i>	productR returns the relational product of two relations A and B in logical (boolean) matrix format	A, B: logical matrices which can represent relations
<i>unionR(A, B)</i>	unionR returns the union of two relations A and B in logical (boolean) matrix format	A, B: logical matrices which can represent relations
<i>intersectionR(A, B)</i>	intersectionR returns the intersection of two matrices A and B in logical (boolean) matrix format	A, B: logical matrices which can represent relations
<i>differenceR(A, B)</i>	differenceR returns the difference of two matrices A and B in logical (boolean) matrix format	A, B: logical matrices which can represent relations
<i>complementR(A)</i>	complementR returns the logical complement of a matrix A in logical (boolean) matrix format	A: logical matrix
<i>transposeR(A)</i>	transposeR returns the transpose of matrix A in logical (boolean) matrix format	A: logical matrix

Table 9.3: Zero, One and Identity boolean matrices

<i>Routine</i>	<i>Usage</i>	<i>Parameters</i>
<i>zeroB</i>	generate rectangular boolean matrix with all entries = 0 or false	n, m: Number of rows and columns in zero matrix
<i>oneB</i>	generate rectangular boolean matrix with all entries = 1 or true	n, m: Number of rows and columns in one matrix
<i>identityB</i>	generate square boolean identity matrix with entries on diagonal = 1 or true and rest of entries = 0	n: Number of rows and columns in zero matrix

Chapter 10

Literal String Processing MATLAB Functionality

MATLAB (The Mathworks Inc. [120] and Hanselman and Littlefield [53]) uses a cell array data structure to store and manipulate arrays and matrices containing data representing character strings.

MATLAB function to support literal string processing required for programs described in chapter 12 were developed.

Functions to multiply strings with boolean data as well arrays of strings stored in MATLAB cell arrays were developed.

Refer to Appendix F showing the MATLAB code with an example.

Chapter 11

Engineering Process Model

11.1 Introduction

The use of process models in the analysis, optimisation and simulation of processes has proven to be extremely beneficial in the instances where they could be applied appropriately. However, the Architecture/Engineering/Construction (AEC) industries present unique challenges that complicate the modelling of their processes.

A simple Engineering process model, based on the specification of Tasks, Datasets, Persons and Tools, and certain relations between them, has been developed, and its advantages over conventional techniques have been illustrated by Huhnt [60]. This model is based on the premise that the engineering process concerns itself with the stage-wise development of Datasets. These Datasets may be e.g. drawings, reports, specifications, analysis and design sheets. Persons specifically execute Tasks for the purpose of developing the Datasets. The stages in the evolution of Datasets are given status values, e.g. preliminary, engineered, checked, final. Tools are used to operate on Datasets. The relations between the sets of Tasks, Persons, Datasets and Tools needed to execute the project and structure the model can be used to provide management information. Certain relations need to be defined and the remaining ones are computed using relational algebra operations. The main advantage of this approach is that the complex relations are computed, while only three simple relations are specified.

By mapping Tasks, Datasets, Persons and Tools to vertices, and the relationships between them to edges, directed graphs can be formed. Of special importance is the ‘has to be executed before’ graph in the set of Tasks. If the project schedule adheres to the sequence of Tasks in this graph, consistent evolution of the Datasets is guaranteed. An optimal step schedule is obtained by topologically sorting the graph.

Eygelaar and Van Rooyen [37] addresses the fact that the optimal step schedule solution is complicated by the fact that engineering projects are often executed with limited resources. As a result, resources may be over-utilised in any step of the optimal schedule, and determining the impact of resource restrictions on the step schedule becomes important. Task-shifting strategies are required to find a near-optimal sequence of Tasks that guarantees consistent Dataset evolution while resolving resource restrictions.

In this chapter the basic theory of the Engineering process model is described.

Two examples applicable to the consulting engineering environment are developed in this chapter. In the first example the task step schedule is computed without reference to the dataset development while in the second example the effect of the evolution of datasets on the step schedule is demonstrated.

The material covered in this chapter refers to developments documented by Huhnt et al. [62],[58],[57],[59] and [61] as well as Eygelaar [38], Eygelaar and Van Rooyen [37] and Lawrence [71].

11.2 Engineering process model, components and relations

The process of executing an engineering project is modelled using a relational model. In this section the components of the model, relations and order relations in the set of Tasks and the specification of the process model is dealt with.

11.2.1 Components of the model

Four sets of components have been identified as building blocks for the process model, namely the set of Tasks, the set of Datasets, the set of Persons and the set of Tools. Tasks represent operations on

Datasets, raising the status value of the Dataset in the process. Tasks are executed by Persons and Datasets are operated upon using Tools. There are 16 possible binary relations between the components, of which only 3 are user specified while the remaining 13 relations can be derived mathematically. The three user-specified heterogeneous binary relations are the relation between the set of Tasks and the set of Datasets (Task-Dataset relation), the relation between the set of Tasks and the set of Persons (Task-Person relation), and the relation between the set of Datasets and the set of Tools (Dataset-Tool relation). Refer to figure 11.1 as well as figure 11.2 for an overview of the binary relations and their semantics.

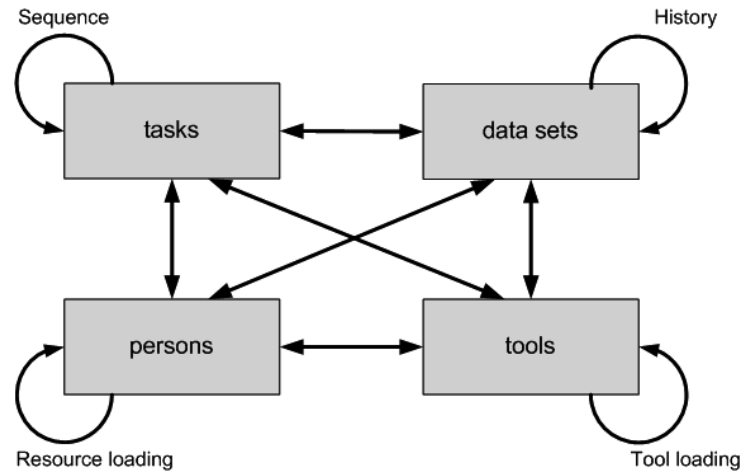


Figure 11.1: Engineering process model relations

	Person	Task	Dataset	Tool
Person		executes	access (read/write)	use
Task	is executed by		access (read/write/ modify)	requires
Dataset	is accessed by (read/write)	is accessed by (read/ write/modify)		can be edited by
Tool	is used by	is required by	can edit	

*Highlighted binary relations are user specified.

Figure 11.2: Overview of engineering process model binary relations

11.2.2 Relations in the set of Tasks

On the diagonal in figure 11.2 there are four homogeneous binary relations. Of these, the relation in the set of Tasks shown in equation 11.2.1 is the most important.

$$\begin{aligned} \text{Relation in the set of Tasks (T)} &:= \\ \{(task_x, task_y) \in T \times T | task_x \neq task_y \wedge task_x \text{ 'has to be executed before' } task_y\} \end{aligned} \quad (11.2.1)$$

The Task-Dataset relation and three predefined rules described in section 11.2.5 are used to determine the relation in equation 11.2.1 in the set of Tasks. This relation is called the Unconstrained Consistent Sequence of Tasks (uCST) since no account is taken of resource limitations during its creation. When regarded as a graph, it is called the uCST graph (Eygelaar [38]). Every edge in the uCST graph represents a relationship between its incident Tasks that must be honoured in order to guarantee consistent development of Datasets.

11.2.3 Step schedule of tasks

The uCST graph is sorted topologically, the result of which is a step schedule. Each step contains the Tasks that have to be executed before the Tasks in the following steps can be executed. The step schedule of the uCST is considered an optimal solution since the least number of steps is used and Tasks are assigned to the earliest possible step.

11.2.4 Relations in the sets of Persons, Tools and Datasets

Homogeneous binary relation	Meaning of relation	Required information to derive relation
Relation in the set of <i>Tasks</i> (<i>uCST-graph</i>)	Consistent sequence of <i>Tasks</i>	<i>Task-Dataset</i> relation and three predefined rules
Relation in the set of <i>Persons</i> (<i>Person-Person</i> relation)	<i>Person</i> loading (Which <i>Person</i> is utilized in what logical step)	<i>uCST-solution</i> and <i>Task-Person</i> relation
Relation in the set of <i>Tools</i> (<i>Tool-Tool</i> relation)	<i>Tool</i> loading (Which <i>Tool</i> is utilized in what logical step)	<i>uCST-solution</i> and <i>Task-Tool</i> relation
Relation in the set of <i>Datasets</i> (<i>Dataset-Dataset</i> relation)	<i>Dataset</i> evolution (What a <i>Dataset's</i> status is at the beginning and end of each logical step)	<i>uCST-solution</i> and <i>Task-Dataset</i> relation

Figure 11.3: Overview of engineering process homogeneous binary relations

The remaining 3 homogeneous binary relations, shown in figure 11.3, can be derived using the uCST solution together with the other heterogeneous binary relations. For example, the relation in the set of Persons can be derived using the uCST graph and the user specified Task-Person relation.

11.2.5 Order relation in the set of Tasks

Three rules were identified which govern the 'has to be executed before' ordering of Tasks in an engineering project. The resulting uCST order relation in the set of Tasks is neither strict nor total. Since the relation is not total, the uCST graph may contain cycles. However, these are simply interpreted as tasks that have to be executed in parallel, and can be represented in the graph by super tasks, thereby obtaining an acyclic graph. Each rule is defined and described below.

Rule 1: A Dataset has to be created before it can be read or modified

It is a basic rule that Datasets have to be created before they can be utilised. For example, if Dataset D is created by $Task_x$, and the same Dataset D is either read or modified by $Task_y$, the relationship $Task_x$

‘has to be executed before’ $Task_y$ is true, i.e. the pair $(Task_x, Task_y)$ is an element of the ‘has to be executed before’ relation. Since a Task can read, create or modify more than one Dataset, the example described above has to be extended to fit generic cases. The rule is set out in equation 11.2.2.

Given:

$\{data_x^{create}\}$ = Set of Datasets created by $Task_x$

$\{data_y^{read}\}$ = Set of Datasets read by $Task_y$

$\{data_y^{modify}\}$ = Set of Datasets modified by $Task_y$

The mathematical representation of Rule 1 is:

(11.2.2)

If:

$\{data_x^{create}\} \cap \{data_y^{read}\} \cup \{data_y^{modify}\} \neq \phi$

Then:

$(Task_x) R (Task_y) = true$

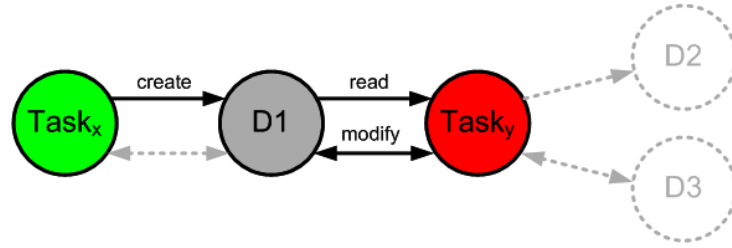


Figure 11.4: Rule 1: Order relation in the set of tasks

Rule 2: The status of data has to increase during modification

Different Tasks can modify the same Dataset and at the conclusion of each Task the Dataset has a certain status, e.g. ‘preliminary’ or ‘engineered’. For example, $Task_x$ modifies Dataset D and increases its status rank to $r(D)_x$. $Task_y$ modifies the same Dataset D and increases its status rank to $r(D)_y$. If $r(D)_x$ is smaller than $r(D)_y$, then the relationship $Task_x$ ‘has to be executed before’ $Task_y$ is true, i.e. the pair $(Task_x, Task_y)$ is an element of the ‘has to be executed before’ relation. Since a Task can modify more than one Dataset, the example described above has to be extended in general. The rule is set out in equation 11.2.3.

Given:

$\{data_x^{modify}\}$ = Set of Datasets modified by $Task_x$

$\{data_y^{modify}\}$ = Set of Datasets modified by $Task_y$

The mathematical representation of Rule 2 is:

(11.2.3)

If:

$p \in \{data_x^{modify}\} \cap \{data_y^{modify}\} \wedge r(p)_x < r(p)_y$

Then:

$(Task_x) R (Task_y) = true$

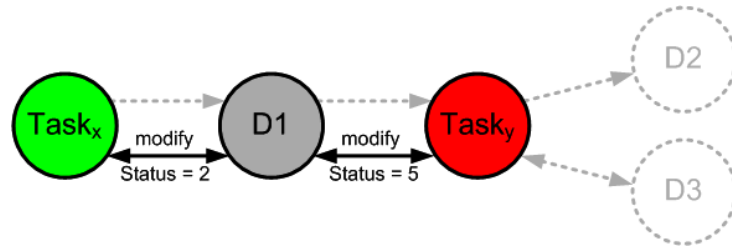


Figure 11.5: Rule 2: Order relation in the set of tasks

Rule 3: For any Task, the highest status rank of its output data cannot be higher than the lowest status rank of its input data

Rule 3 focuses on a Task delivering a set of Datasets which another Task requires as a set of input Datasets. The latter Task generates a disjoint set of output Datasets. The statuses of the set of input Datasets must be at a sufficient level in order to produce the disjoint set of output Datasets at specific status levels. For example, the following status ranking is available: assumed (1) > engineered (2) > final (3) (low to high). $Task_x$ modifies Dataset D_1 to status level engineered. $Task_y$ reads Dataset D_1 , and creates a different Dataset D_2 at a status level of assumed and modify another different Dataset D_3 to a status level of final. Thus, $Task_x$ ‘has to be executed before’ $Task_y$ to ensure that the minimum status level of input Datasets has already been brought up to an equal status level compared to the maximum status level of output Datasets. Rule 3 is set out in equation 11.2.4.

Given:

$\{data_x^{modify}\}$ = Set of Datasets modified by $Task_x$

$\{data_y^{create}\}$ = Set of Datasets created by $Task_y$

$\{data_y^{read}\}$ = Set of Datasets read by $Task_y$

$\{data_y^{modify}\}$ = Set of Datasets modified by $Task_y$

The mathematical representation of Rule 3 is:

If:

$INPUT = \{Datasets\} = \{data_y^{modify}\} \cap \{data_y^{read}\}$

$OUTPUT = \{Datasets\} = \{data_y^{create}\} \cup \{data_y^{modify}\}$

$r(min)_{INPUT}$ = Minimum status rank in INPUT

$r(max)_{OUTPUT}$ = Minimum status rank in OUTPUT

And if:

$r(min)_{INPUT} < r(max)_{OUTPUT}$ Then: $(Task_x) R (Task_y) = true$

(11.2.4)

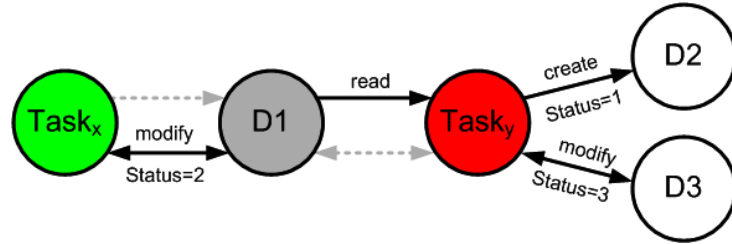


Figure 11.6: Rule 3: Order relation in the set of tasks

11.3 Specification of the process model

The three binary relations that have to be specified by the user are shown in figure 11.2. In order to perform these relationship specifications, it is clear that the complete sets of Tasks, Persons, Datasets and Tools have to be specified beforehand, as well as the list of status values through which Datasets evolve. Depending on the outputs and functionality required from the model, other input data may be required. For example, if the user wants to be able to sort and search the set of Tasks according to certain attributes of the Tasks, the attributes have to be specified and assigned to Tasks as applicable. The user specification of relationships between Tasks, Datasets, Persons and Tools is discussed below.

11.3.1 Task-Dataset relationships

Datasets are produced or operated upon during the execution of Tasks. Three different types of heterogeneous binary relationships can be identified between a Task and a Dataset: a Dataset can be read by a Task, a Dataset can be modified by a Task or a Dataset can be created by a Task. A Task cannot delete a Dataset since records must always be available for future reference. Multiple Task-Dataset relationships can be assigned to a Task or to a Dataset object, but the same Task cannot operate on the same Dataset more than once.

The relationships described below are shown in diagrammatic form in figure 11.7

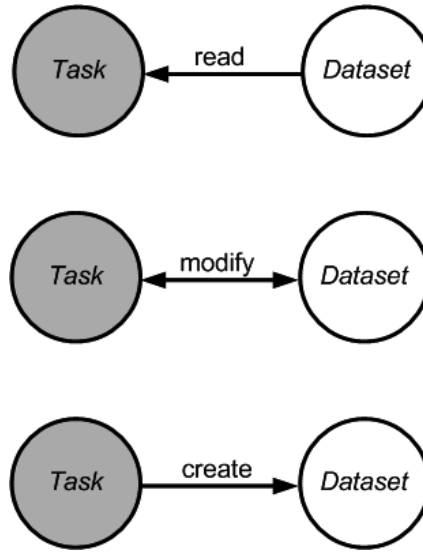


Figure 11.7: Task-Dataset relationships: Read, modify, create

Create

A ‘create’ heterogeneous binary relationship between a Task and a Dataset indicates that during execution the Task is creating the Dataset. Not all Datasets need to be created by Tasks in the same process model as Datasets may already be in existence. Not all Tasks have to create Datasets. Datasets are created at a specific status level.

Read

A ‘read’ relationship between a Task and a Dataset indicates that during its execution the Task is reading the Dataset. It is not compulsory that each Task must read a Dataset or that each Dataset must be read by a Task.

Modify

A ‘modify’ relationship between a Task and a Dataset indicates that during its execution the Task is modifying the Dataset. During modification a Task actually reads and overwrites the same Dataset, but it is considered a single relationship. The status level of the Dataset is increased by the modification.

11.3.2 Task-Person relationships

The responsibility of a Person to execute a Task is assigned by this relationship. Even if the Task is automated, the responsibility must still be assigned. Thus, a Task must always be assigned at least one Person with the possibility that multiple Persons might be assigned to the same Task, however it is not allowed to assign the same Person to the same Task more than once. It is important to note that only the responsibility of Task execution is assigned by a Task-Person relationship and that assigning more Persons to a Task will not reduce the execution duration of the Task.

11.3.3 Dataset-Tool relationships

Datasets are read, modified or created using a Tool. A Tool can take on many different forms ranging from CAD software to a drawing board. The relationship should be read as a Dataset ‘is edited by’ a Tool. When this relationship is specified a Tool operates on a Dataset. At least one Dataset-Tool relationship should be specified for each Dataset. Multiple Tools can be assigned to a Dataset; however it is not allowed to assign the same Tool to the same Dataset more than once. It is important to note

that the Dataset-Tool relationship only assigns which Tools operate on a Dataset and that assigning more Tools to a Dataset will not reduce the execution duration of the Tasks operating on the Dataset.

In tabular form the conceptual model can be viewed as shown in table 11.1.

A directed graph (see section 5.6) is suitable for describing relationships between the elements of a set such as the ‘has to be executed before’ relation in the set of tasks. The task elements of the set are called vertices of the graph and are identified by their labels.

The model is set up as binary and unary relations using boolean adjacency matrices.

Not all relations need to be specified seeing that basic operations on the relations can be performed to generate the other relations.

The relations which need to be specified as input for determining other relations are:

- which tasks are executed by which persons
- which tasks create which units of data
- which tasks read which units of data
- which tasks modify which units of data
- which units of data are edited by which tools

11.4 Example A: Consulting engineering business process model

A complete example drawn from the consulting engineering service business environment is given below. A typical building project is modelled at a high level.

The example was programmed in MATLAB [53] and the graphical output processed using the yEd [130] visualisation software for directed graphs.

The relations which are specified are listed in table 11.2.

The example engineering process model uses the following objects:

Persons defined:

Client or owner
 Architect
 Engineer
 Quantity surveyor
 Constructor, Contractor or Builder

Tasks identified:

Conceptualise project
 Planning
 Engineering design
 Specification and documentation
 Quantity take-off
 Construction and building

Tools used (software):

Text processor or word processor
 Computer Aided Design (CAD) software
 Engineering design software
 Quantities take-off and processing software
 Construction process planning software

Datasets selected:

Concept drawings

Architects drawings
 Design calculations
 Engineering drawings
 Specifications
 Bill of quantities
 Construction programmes and schedules
 As-built drawings

Boolean adjacency matrices are used to specify the relations between the sets above. The data is listed in tables 11.3, 11.4, 11.5, 11.6, 11.7 and 11.8

- The relation specified between persons and tasks i.e. person executes task is shown in table 11.3.
- The relation specified between tasks and datasets created by tasks is shown in table 11.4
- The relation specified between tasks and datasets read by tasks is shown in table 11.5.
- The relation specified between tasks and datasets modified by tasks is given in table 11.6.
- The relation specified between datasets and tools required to create / read / modify datasets is shown in table 11.7.
- The relation specified between tasks and tools used to execute tasks is given in 11.8.

An overview of the relations specified is given below in a set of diagrams in directed bipartite graph format. Refer to figures 11.8, 11.9, 11.10, 11.11 and 11.12.

Figure 11.13 shows the relation ‘task uses tool’ which is specified here but can be computed later.

11.5 Relations computed from specified process model relations

Derived relations describing logical links between the elements of the sets of persons, tasks, tools and datasets can now be computed using relational algebra operations. The relational operations performed on the process model (boolean relations) are set out below.

11.5.1 Relations deduced by transposing the specified relations

Relations which are computed by transposing the relations specified in tables 11.3, 11.4, 11.5, 11.6, 11.7 and 11.8 are shown in equation 11.5.1.

$$\begin{aligned}
 \mathbf{R}_{Person-Task} &= \mathbf{R}_{Task-Person}^T \\
 \text{and} \\
 \mathbf{R}_{DataCreate-Task} &= \mathbf{R}_{Task-DataCreate}^T \\
 \mathbf{R}_{DataRead-Task} &= \mathbf{R}_{Task-DataRead}^T \\
 \mathbf{R}_{DataModify-Task} &= \mathbf{R}_{Task-DataModify}^T \\
 \text{and} \\
 \mathbf{R}_{Tool-Dataset} &= \mathbf{R}_{Dataset-Tool}^T
 \end{aligned} \tag{11.5.1}$$

Bipartite directed graph format diagrams of the relations computed as specified are shown in figures 11.14, 11.15, 11.16 and 11.17.

11.5.2 Relations deduced by forming the union of all three Dataset-task relations

Further task-data relations deduced by forming the union of all three Dataset-task relations - create / read / modify. The task operates on dataset relation can then be formed by transposing the combined data operated on by task relation. Refer to equation 11.5.2.

$$\begin{aligned}
 \mathbf{R}_{Data-Task} &= \mathbf{R}_{DataCreate-Task} \cup \mathbf{R}_{DataRead-Task} \cup \mathbf{R}_{DataModify-Task} \\
 \mathbf{R}_{Task-Data} &= \mathbf{R}_{Task-DataCreate} \cup \mathbf{R}_{Task-DataRead} \cup \mathbf{R}_{Task-DataModify} \\
 \mathbf{R}_{Task-Data} &= \mathbf{R}_{Data-Task}^T
 \end{aligned} \tag{11.5.2}$$

Bipartite directed graph format diagrams of the relations computed as specified are shown in figures 11.18 and 11.19.

Table 11.1: Engineering process model concept

	<i>Persons</i>	<i>Tasks</i>	<i>Datasets</i>	<i>Tools</i>
<i>Persons</i>	resource loading	execute	create, modify read	use
<i>Tasks</i>	are executed by	sequence	create modify read	require
<i>Datasets</i>	are created modi- fied read by	are created modi- fied read by	history	are modified by (created / read)
<i>Tools</i>	are used by	are required for	modify (create / read)	tool loading

Table 11.2: Relations specified for model example in consulting engineering process

<i>Relation from set</i>	<i>Relation to set</i>	<i>Description</i>
Persons / Groups	Tasks	Person executes task
Tasks	Dataset	Task creates dataset
Tasks	Dataset	Task reads dataset
Tasks	Dataset	Task modifies dataset
Task	Tools	Task uses tool

Table 11.3: Boolean adjacency matrix representation of relation Persons executes Task $\mathbf{R}_{Person-Task}$

<i>Task:</i>	<i>Conceptualise</i>	<i>Planning</i>	<i>Engineering</i>	<i>Specify and document</i>	<i>Quantity take-off</i>	<i>Construction and building</i>
<i>Person:</i>						
Client / Owner	1					
Architect	1	1				
Engineer			1	1		
Quantity Sur- veyor				1	1	
Constructor / Contractor / Builder						1

Table 11.4: Boolean adjacency matrix representation of relation Task creates Dataset $\mathbf{R}_{Task-DataCreate}$

<i>Datasets:</i>	<i>Concept drawings</i>	<i>Architects drawings</i>	<i>Engineering design cal- culations</i>	<i>Engineering drawings</i>	<i>Specifications</i>	<i>Bill of quantities</i>	<i>Construction Pro- grammes Schedules</i>	<i>As-built drawings</i>
<i>Task:</i>								
Conceptualise	1							
Planning		1						
Engineering			1	1				
Specify and document					1			
Quantity take-off						1		
Construction and build- ing							1	1

Table 11.5: Boolean adjacency matrix representation of relation Task reads Dataset $\mathbf{R}_{Task-DataRead}$

<i>Datasets:</i>	<i>Concept drawings</i>	<i>Architects drawings</i>	<i>Engineering design calculations</i>	<i>Engineering drawings</i>	<i>Specifications</i>	<i>Bill of quantities</i>	<i>Construction Pro-grammes Schedules</i>	<i>As-built drawings</i>
<i>Task:</i>								
Conceptualise								
Planning	1	1						
Engineering	1	1						
Specify and document		1	1					
Quantity take-off		1		1	1			
Construction and building		1		1	1	1	1	

Table 11.6: Boolean adjacency matrix representation of relation Task modifies Dataset $\mathbf{R}_{Task-DataModify}$

<i>Datasets:</i>	<i>Concept drawings</i>	<i>Architects drawings</i>	<i>Engineering design calculations</i>	<i>Engineering drawings</i>	<i>Specifications</i>	<i>Bill of quantities</i>	<i>Construction Pro-grammes Schedules</i>	<i>As-built drawings</i>
<i>Task:</i>								
Conceptualise	1	1						
Planning		1						
Engineering			1	1				
Specify and document					1			
Quantity take-off						1		
Construction and building							1	1

Table 11.7: Boolean adjacency matrix representation of relation Dataset operated on by Tool $\mathbf{R}_{Data-Tool}$

<i>Tools:</i>	<i>Text processor</i>	<i>CAD software</i>	<i>Engineering design software</i>	<i>Quantities software</i>	<i>Construction planning software</i>
<i>Datasets:</i>					
Concept drawings	1	1			
Architects drawings		1			
Design calculations			1		
Engineering drawings		1	.		
Specifications	1	.	.		
Bill of quantities		.	.	1	
Construction programmes & schedules		.	.		1
As-built drawings		1	.		

Table 11.8: Boolean adjacency matrix representation of relation Task uses Tool $\mathbf{R}_{Task-Tool}$

<i>Tools:</i>	<i>Text processor</i>	<i>CAD software</i>	<i>Engineering design software</i>	<i>Quantities software</i>	<i>Construction planning software</i>
<i>Task:</i>					
Conceptualise	1				
Planning	1	1			
Engineering	1	1	1		
Specify and document	1				
Quantity take-off				1	
Construction and building				.	1

11.5.3 Relations computed between persons and datasets

Deduced relations between persons and datasets can also be computed.

Relations computed by forming relational products of specified / determined relations

Relations which can be computed from the given and deduced relations linking persons and datasets are given in equation 11.5.3.

$$\begin{aligned}
 \mathbf{R}_{Person-DataCreate} &= \mathbf{R}_{Task-Person}^T \bullet \mathbf{R}_{DataCreate-Person}^T \\
 \text{or} \\
 \mathbf{R}_{Person-DataCreate} &= \mathbf{R}_{Person-Task} \bullet \mathbf{R}_{Task-DataCreate} \\
 \text{and} \\
 \mathbf{R}_{Person-DataRead} &= \mathbf{R}_{Task-Person}^T \bullet \mathbf{R}_{DataRead-Person}^T \\
 \text{or} \\
 \mathbf{R}_{Person-DataRead} &= \mathbf{R}_{Person-Task} \bullet \mathbf{R}_{Task-DataRead} \\
 \text{and} \\
 \mathbf{R}_{Person-DataModify} &= \mathbf{R}_{Task-Person}^T \bullet \mathbf{R}_{DataModify-Person}^T \\
 \text{or} \\
 \mathbf{R}_{Person-DataModify} &= \mathbf{R}_{Person-Task} \bullet \mathbf{R}_{Task-DataModify}
 \end{aligned} \tag{11.5.3}$$

The outcome of these operations in directed graph format diagrams are shown in figures 11.20, 11.21 and 11.22.

Relations computed by transposing the computed person-data relations

The person-dataset relations computed above can be transposed as shown in equation 11.5.4.

$$\begin{aligned}
 \mathbf{R}_{DataCreate-Person} &= \mathbf{R}_{Person-DataCreate}^T \\
 \text{and} \\
 \mathbf{R}_{DataRead-Person} &= \mathbf{R}_{Person-DataRead}^T \\
 \text{and} \\
 \mathbf{R}_{DataModify-Person} &= \mathbf{R}_{Person-DataModify}^T
 \end{aligned} \tag{11.5.4}$$

11.5.4 Relations deduced by forming the union of the person - data and data - person relations

Relations deduced by forming the union of the person-data(create/read/modify) and data(create/read/modify)-person relations are shown in equation 11.5.5 For demonstration purposes the transpose of the first union is also shown which leads to the same results. Figure 11.21 shows a graph format view of the result.

$$\begin{aligned}
 \mathbf{R}_{Person-Data} &= \mathbf{R}_{Person-DataCreate} \cup \mathbf{R}_{Person-DataRead} \cup \mathbf{R}_{Person-DataModify} \\
 \mathbf{R}_{Data-Person} &= \mathbf{R}_{DataCreate-Person} \cup \mathbf{R}_{DataRead-Person} \cup \mathbf{R}_{DataModify-Person} \\
 \mathbf{R}_{Data-Person} &= \mathbf{R}_{Person-Data}^T
 \end{aligned} \tag{11.5.5}$$

11.5.5 Relations computed using relations specified between persons and tools used by persons

Equation 11.5.6 shows the person-tool and tool-person relations which can be computed.

$$\begin{aligned}
 \mathbf{R}_{Person-Tool} &= (\mathbf{R}_{Person-DataCreate} \cup \mathbf{R}_{Person-DataRead} \cup \mathbf{R}_{Person-DataModify}) \bullet \mathbf{R}_{Data-tool} \\
 \mathbf{R}_{Tool-Person} &= \mathbf{R}_{Person-Tool}^T
 \end{aligned} \tag{11.5.6}$$

Figures 11.23 and 11.24 depict the resulting relations in directed graph format.

11.5.6 Relations computed using relations specified between tasks and tools used to execute tasks

Relations computed using relations specified between tasks and tools used to execute tasks are shown in equation 11.5.7.

$$\begin{aligned} \mathbf{R}_{Task-Tool} &= (\mathbf{R}_{Task-DataCreate} \cup \mathbf{R}_{Task-DataRead} \cup \mathbf{R}_{Task-DataModify}) \bullet \mathbf{R}_{Data-tool} \\ \mathbf{R}_{Tool-Task} &= \mathbf{R}_{Task-Tool}^T \end{aligned} \quad (11.5.7)$$

$$\begin{aligned} \mathbf{R}_{Tool-Task} &= \mathbf{R}_{Task-Tool}^T \\ \mathbf{R}_{Person-Tool} &= \mathbf{R}_{Person-Task} \bullet \mathbf{R}_{Task-Tool} \\ \text{and} \\ \mathbf{R}_{Tool-Person} &= \mathbf{R}_{Person-Tool}^T \\ \mathbf{R}_{Tool-DataCreate\&Read\&Modify} &= \mathbf{R}_{DataCreate\&Read\&Modify-Tool}^T \end{aligned} \quad (11.5.8)$$

$$\begin{aligned} \mathbf{R}_{DataCreate-Tool} &= \mathbf{R}_{DataCreate-Task} \bullet \mathbf{R}_{Task-Tool} \\ \mathbf{R}_{DataRead-Tool} &= \mathbf{R}_{DataRead-Task} \bullet \mathbf{R}_{Task-Tool} \\ \mathbf{R}_{DataModify-Tool} &= \mathbf{R}_{DataModify-Task} \bullet \mathbf{R}_{Task-Tool} \\ \mathbf{R}_{Data-Tool} &= \mathbf{R}_{DataCreate-Tool} \cup \mathbf{R}_{DataRead-Tool} \cup \mathbf{R}_{DataModify-Tool} \\ \mathbf{R}_{Task-Tool} &= (\mathbf{R}_{Task-DataCreate} \cup \mathbf{R}_{Task-DataRead} \cup \mathbf{R}_{Task-DataModify}) \bullet \mathbf{R}_{Data-Tool} \end{aligned} \quad (11.5.9)$$

Graph format representations of the relations computed are given in figure 11.13.

11.5.7 Relations deduced from dataset requires tool relation

Typical relation products to deduce other relations via tools are not useful as such because tools have multiple relations with datasets are shown in equation 11.5.10.

$$\begin{aligned} \mathbf{R}_{Person-Dataset} &= \mathbf{R}_{Person-Tool} \bullet \mathbf{R}_{Tool-Dataset} \\ \mathbf{R}_{Dataset-Task} &= \mathbf{R}_{Dataset-Tool} \bullet \mathbf{R}_{Tool-Task} \\ \mathbf{R}_{Task-Dataset} &= \mathbf{R}_{Task-Tool} \bullet \mathbf{R}_{Tool-Dataset} \end{aligned} \quad (11.5.10)$$

The graph format diagrams of the computed relations are shown in figures 11.26, 11.27 and 11.28.

11.5.8 Computing the logical sequence of tasks

Equation 11.5.11 was used to compute the logical sequence of tasks based on dataset creation, read and modification. The intermediate result as well as the logical sequence of tasks computed are shown in figures 11.29 and 11.30.

It is interesting to note that each task carries a logical link to itself in the computed relation.

$$\mathbf{R}_{Task-Task} = \mathbf{R}_{DataCreate-Task} \bullet (\mathbf{R}_{DataRead-Task} \cup (\mathbf{R}_{DataModify-Task})) \quad (11.5.11)$$

11.5.9 Computing person loading

Person loading i.e. the sequence in which persons are required, is computed using the data create/read and modify relationship shown in equation 11.5.12.

$$\mathbf{R}_{Person-Person} = \mathbf{R}_{DataCreate-Person} \bullet (\mathbf{R}_{DataRead-Person} \cup (\mathbf{R}_{DataModify-Person})) \quad (11.5.12)$$

11.5.10 Computing tool loading

The tool loading computes tools or tools which need to available with another tool. Refer to equation 11.5.13.

$$\begin{aligned} \mathbf{R}_{Tool-DataCreate} &= \mathbf{R}_{Tool-Data} \\ \mathbf{R}_{DataRead-Tool} &= \mathbf{R}_{DataModify-Tool} = \mathbf{R}_{Tool-Data} \\ \mathbf{R}_{Tool-Tool} &= \mathbf{R}_{Tool-DataCreate} \bullet (\mathbf{R}_{DataRead-Tool} \cup \mathbf{R}_{DataModify-Tool}) \end{aligned} \quad (11.5.13)$$

Figure 11.34 shows the tool loading relation in graph format.

11.5.11 Computing dataset history

The dataset history relation can be computed from the dataset-persons as well as the dataset-tasks relations.

Dataset history via persons - read only

Refer to figure 11.35 for a graphical representation of the logic computed using equation 11.5.14.

$$\mathbf{R}_{Data-Data} = (\mathbf{R}_{DataCreate-Person} \bullet \mathbf{R}_{DataRead-Person})^T \quad (11.5.14)$$

Dataset history via persons - read and modify

Refer to figures 11.36 and 11.37 for a graphical representation of the logic computed using equation 11.5.15.

$$\begin{aligned} \mathbf{R}_{Person-DataRead\&Modify} &= (\mathbf{R}_{Person-DataRead} \cup \mathbf{R}_{Person-DataModify}) \\ \mathbf{R}_{Data-Data} &= (\mathbf{R}_{DataCreate-Person} \bullet (\mathbf{R}_{Person-DataRead} \cup \mathbf{R}_{Person-DataModify}))^T \end{aligned} \quad (11.5.15)$$

Dataset history via tasks - read only

Two alternatives are considered here. For the first alternative the data-data history is determined via the data create-task and data read-task logic as shown in equation 11.5.16 and 11.38.

$$\mathbf{R}_{Data-Data} = (\mathbf{R}_{DataCreate-Task} \bullet \mathbf{R}_{DataRead-Task})^T \quad (11.5.16)$$

For the second alternative read and modify task-data relationships are combined.

Refer to figure 11.39 and 11.40 for a graphical representation of the logic computed using equation 11.5.17.

$$\begin{aligned} \mathbf{R}_{Task-DataRead\&Modify} &= (\mathbf{R}_{Task-DataRead} \cup \mathbf{R}_{Task-DataModify}) \\ \mathbf{R}_{Data-Data} &= (\mathbf{R}_{DataCreate-Task} \bullet (\mathbf{R}_{Task-DataRead} \cup \mathbf{R}_{Task-DataModify}))^T \end{aligned} \quad (11.5.17)$$

11.6 Process task specification reporting for the process model

An algorithm was developed to extract the task specification for each person or group of persons from the process model specification.

An empty square adjacency matrix representing the process-task specification graph is set up consisting of persons and tasks as well as datasets as vertices.

The person executes task Boolean adjacency matrix is processed row wise person by person.

For each person-task relation an edge is added to the process task specification graph which has all the persons/tasks/datasets as vertices.

For each row each task is selected and the dataset rows from the task-data adjacency matrices processed for the create/read and modify logic.

The active datasets are identified and edges added as shown in the MATLAB code in Appendix G.

For each task-data create logical link a single edge is added to the task specification graph.

```
{PersonTaskDataMatrix(iRow,iCol)=dataCreate(iData);
```

For each task-data read logical link a single edge is added in the transposed position to the task specification graph.

```
{PersonTaskDataMatrix(iCol,iRow)=dataRead(iData);
```

For each task-data modify logical link two edges are added to the task specification graph.

```
PersonTaskDataMatrix(iCol,iRow)=dataModify(iData);
```

```
PersonTaskDataMatrix(iRow,iCol)=dataModify(iData);
```

Vertex labels are added where entries are made and the final processing to generate the yEd graph output (.tgf) file only uses vertices which have labels assigned.

The graph format results of the processing is shown in figures 11.41, 11.42, 11.43, 11.44 and 11.45.

The MATLAB code for the generation of the person-task and person-data relations is contained in Appendix H.

11.7 Example B: Data evolution status value processing for process model

The example contained in this section demonstrates the inclusion use of data status values in the engineering process model. Project progress measurement is based on the status (quality level) of data described by a status attribute.

Each dataset contains a ordered list of statuses through which it progresses as the project progresses. A number of the status lists can be maintained and linked to datasets as required.

An example of a data status list is shown in table 11.9.

Table 11.9: Data status values

Data status value	Description
Assumed	Data values based on assumptions
Preliminary	Input from other role players included
Engineered	Data used in design reports - the final concept has been designed
Checked	Approved by client and/or public authority

The data status list are also referred to as data evolution profiles.

To determine the logical sequence of tasks, additional rules are defined which are included in the algorithm which determines the logical sequence of tasks. This is an implementation of the theory developed in section 11.2.5.

The rules are:

Rule 1: Data can only be read or modified after it has been created

Rule 2: If a task modifies data its status in the data status hierarchy must be increased

Rule 3: The status level (quality) of data output (created) by a task cannot exceed that of the data which has been input for a task. In the case of data only being modified by a task, apply *Rule 2*.

The example set out in the sections below demonstrates this additional functionality of the model. Only the graphical output is shown. The MATLAB code in listed in Appendix I.

11.7.1 Process model set specification

Tasks

t1: Architectural Layout Concept

t2: Architectural Design Detail

t3: Architectural Design Check

t4: Structural Layout Concept

t5: Structural Design Detail

t6: Structural Design Check

t7: Concrete Layout Concept

t8: Concrete Design Detail

t9: Concrete Design Check

Persons

- p1: Architect
- p2: Client
- p3: Structural Engineer
- p4: Technologist
- p5: Checking Engineer

Datasets

- d1: Concept Plan
- d2: Architectural Drawings
- d3: Structural Drawings
- d4: Concrete Drawings

Tools

- g1: CAD
- g2: Engineering Calculations
- g3: Spreadsheet
- g4: Word Processor

Dataset status values

- s1: Preliminary/Concept
- s2: Designed/Engineered
- s3: Finalised
- s4: Checked

11.7.2 Person- Task relation specification

The person-task relation is shown in figure 11.46.

11.7.3 Task - Data specification

The task-data create relation is shown in figure 11.47.

The task-data relations for reading and modification of data is shown in figure 11.47.

11.7.4 Computed basic relations

The basic computed relations linking data and tools and persons with tools is shown in figures 11.50 and 11.51 respectively.

The computed relation linking persons and data is shown in figure 11.52 and the relation linking tasks and tools is shown in figure 11.53

11.7.5 Computing task sequence

The task logical sequences are computed according to rules 1, 2 and 3.

The resulting task sequence computed is combined with the previous sequence computed.

Applying Rule 1

The task sequence with rule 1 applied is shown in figure 11.54.

Apply rule 2 & generating union with rule 1

The task sequence with rule 2 applied is shown in figure 11.55. When rule 1 and rule 2 are applied the sequence takes the form shown in figure 11.56

Apply rule 3 & generating union with rule 1 & 2

Figure 11.57 shows the final logical sequence of tasks which in this case does not differ from the previous one.

Step schedule

The sequence of tasks (figure 11.57) can be displayed in a logical step referenced format computed as shown in figure 11.58.

11.7.6 MATLAB implementation of process model with status settings

The complete MATLAB code used to do the relational computations and generate the graphical output is contained in Appendix G.6.

11.7.7 MATLAB code for Engineering Process Model Example

The MATLAB code developed to generate the relations and output in graphical format set out in the previous sections is listed in Appendix G section G.1. The MATLAB code and output is generated using the publish to L^AT_EX facility available in the MATLAB 2006 [120] version.

Use is made of the MATLAB subprograms developed previously listed in Appendices B, C and discussed in chapter 9.

11.8 Data file formats

The data file formats used for yEd [130] graphical data as well as the comma separated data format used to transfer data to the database is shown in Appendix G section G.4.

11.9 Engineering Process Model - Figures

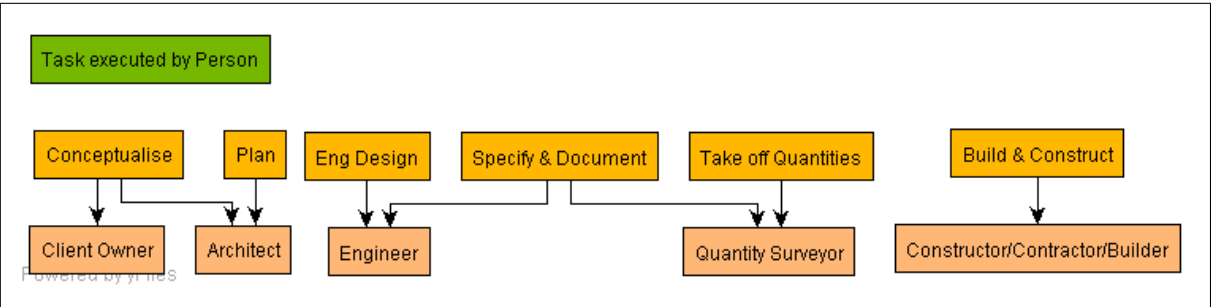


Figure 11.8: Task executed by Person

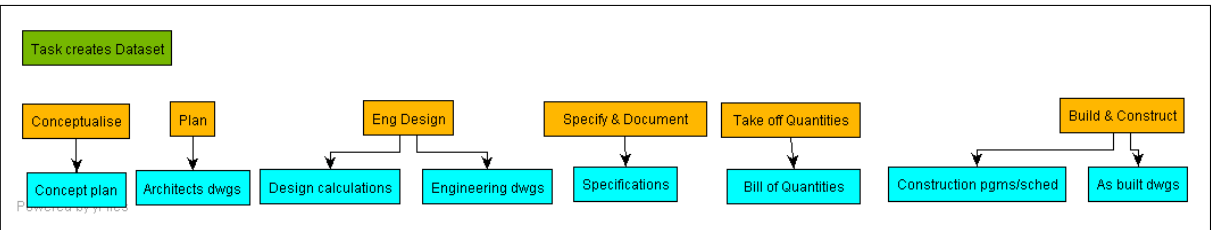


Figure 11.9: Task creates dataset

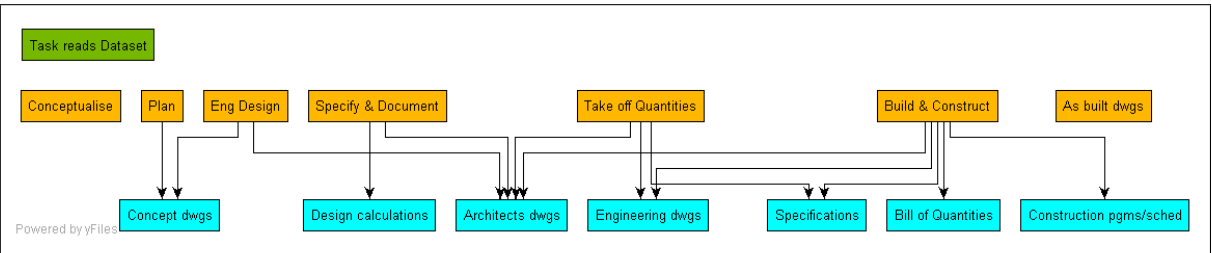


Figure 11.10: Task reads dataset

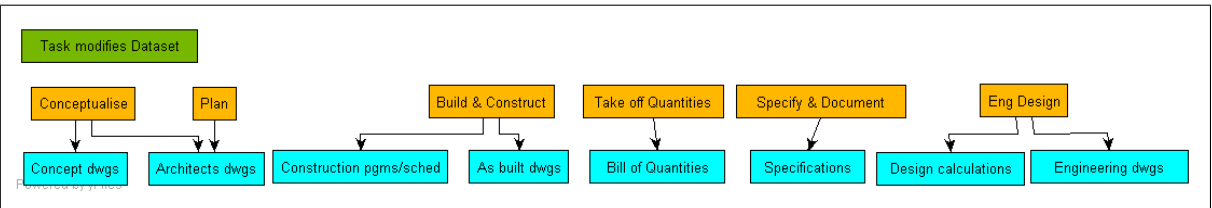


Figure 11.11: Task modifies dataset

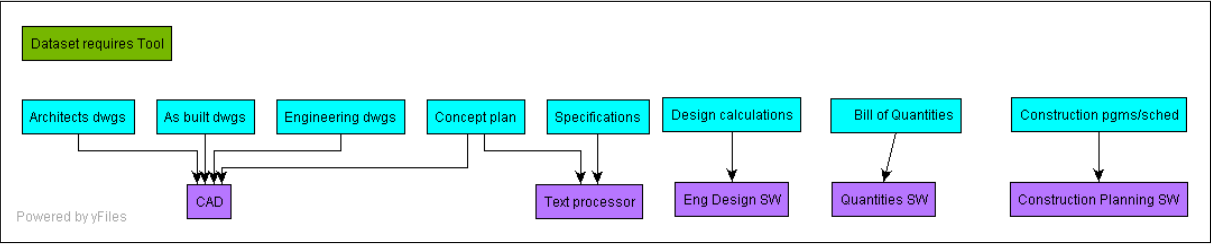


Figure 11.12: Dataset requires tool

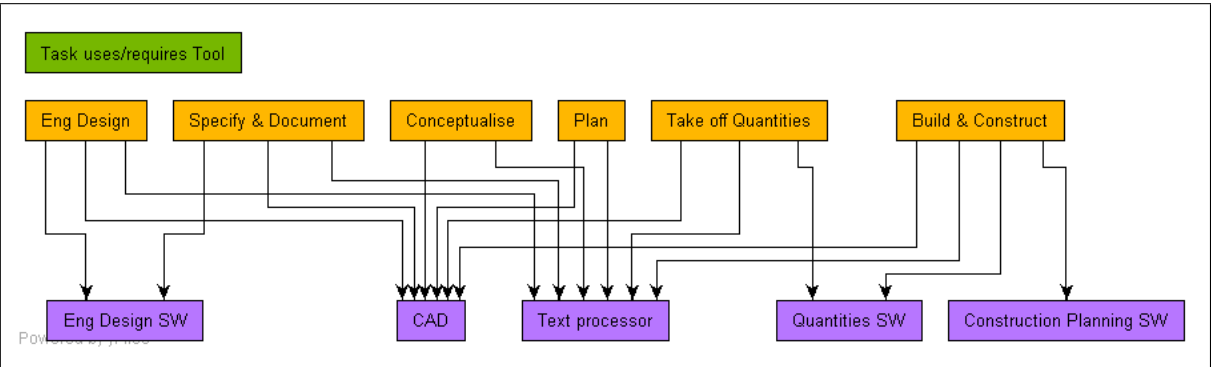


Figure 11.13: Task uses tool

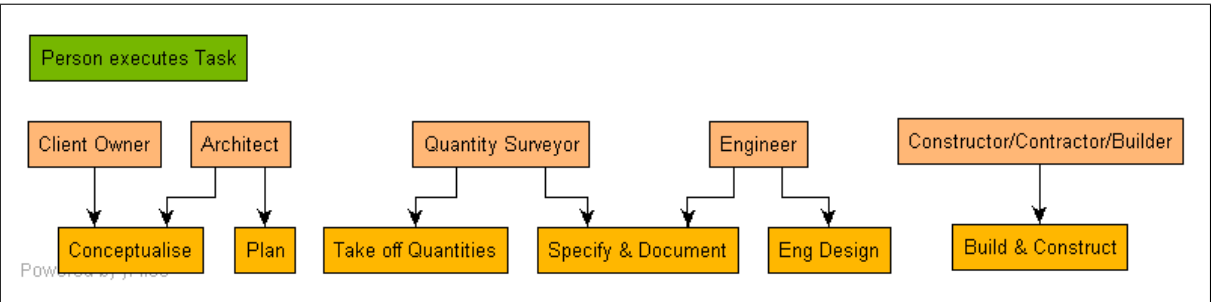


Figure 11.14: Person executes task

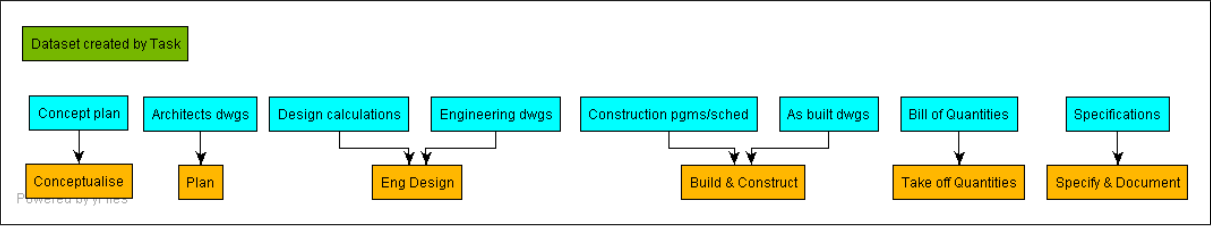


Figure 11.15: Dataset created by Task

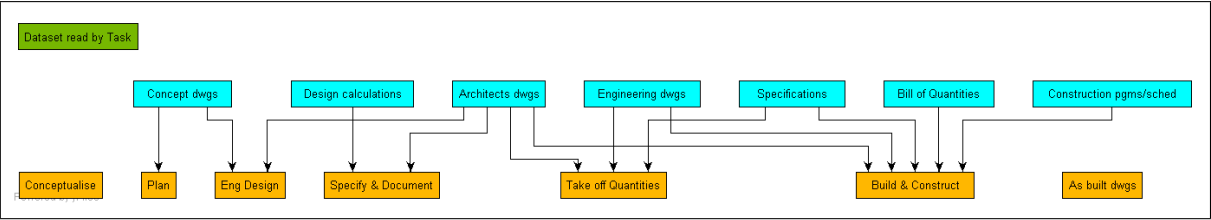


Figure 11.16: Dataset read by Task

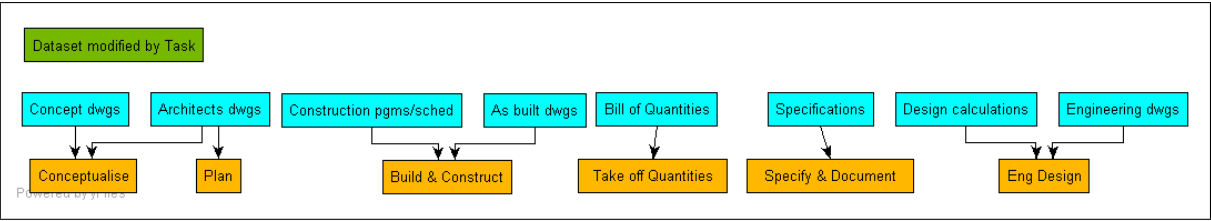


Figure 11.17: Dataset modified by Task

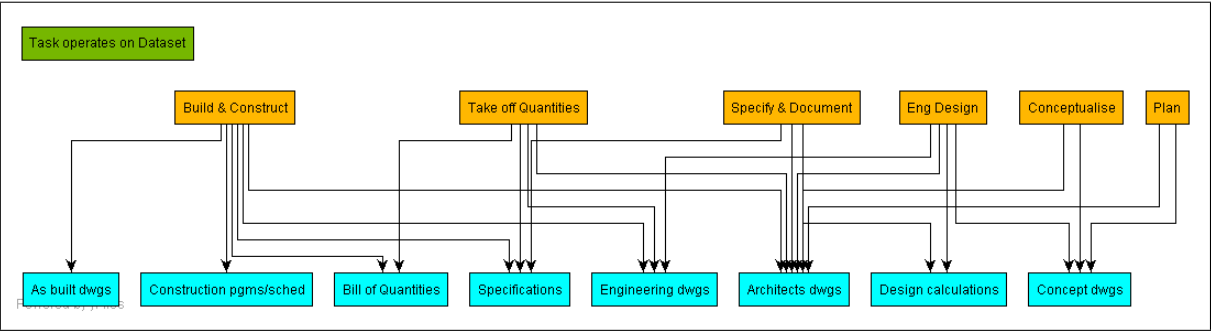


Figure 11.18: Tool operates on dataset

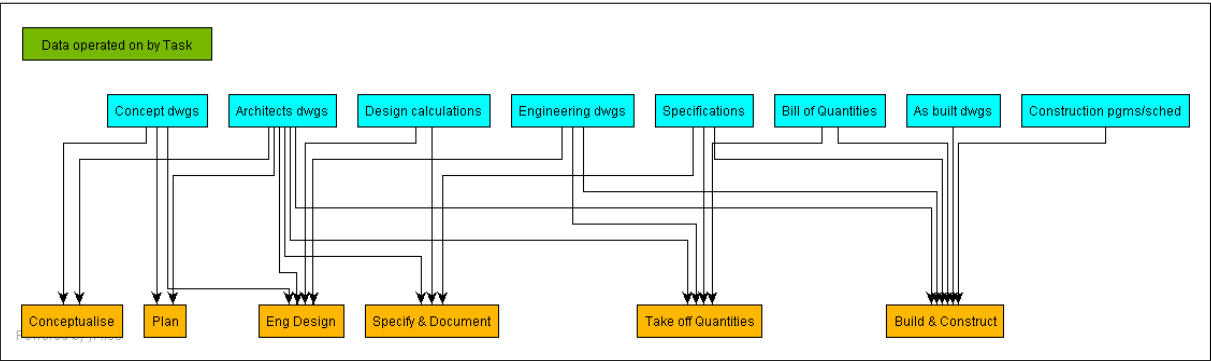


Figure 11.19: Tool operates on dataset

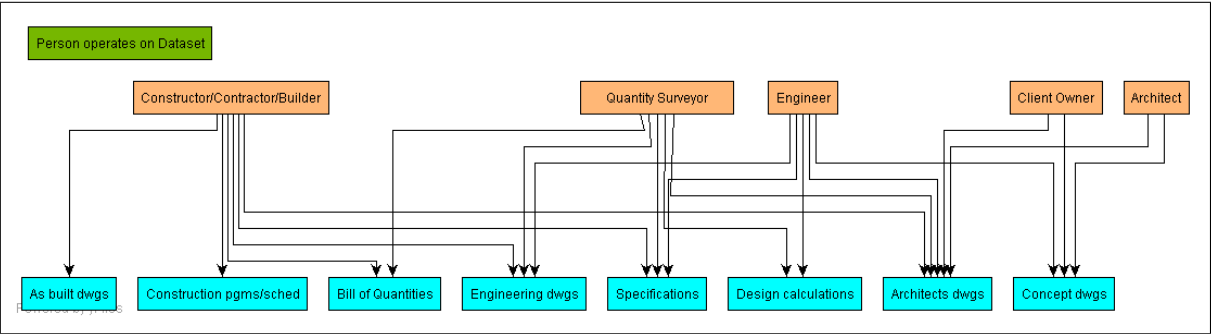


Figure 11.20: *Person operates on dataset*

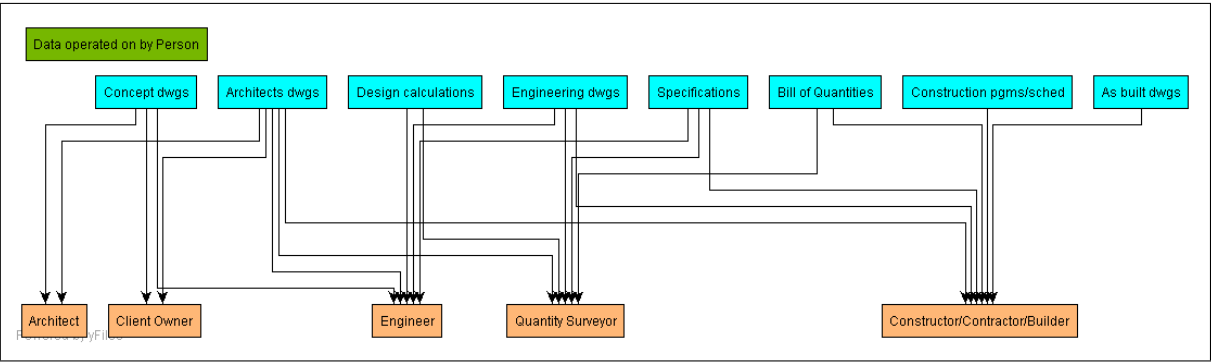


Figure 11.21: *Data operated on by person*

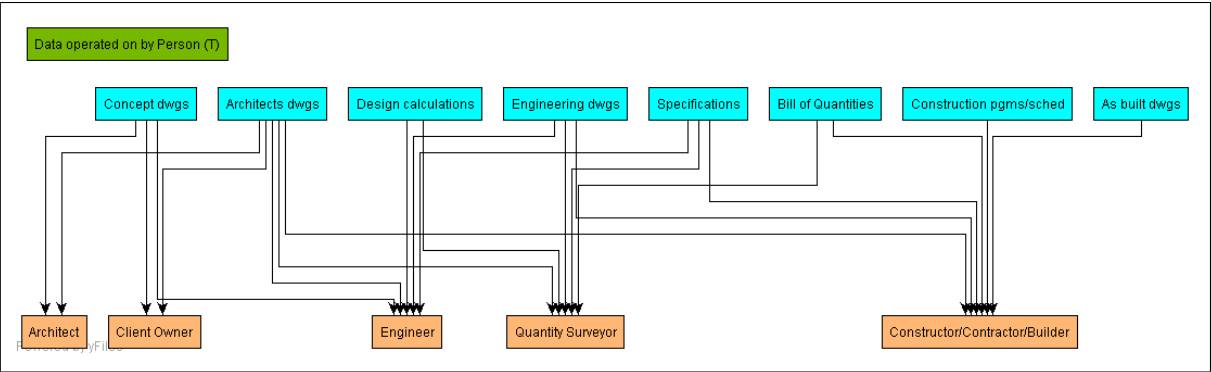


Figure 11.22: *Data operated on by person transposed relation from person operates on dataset*

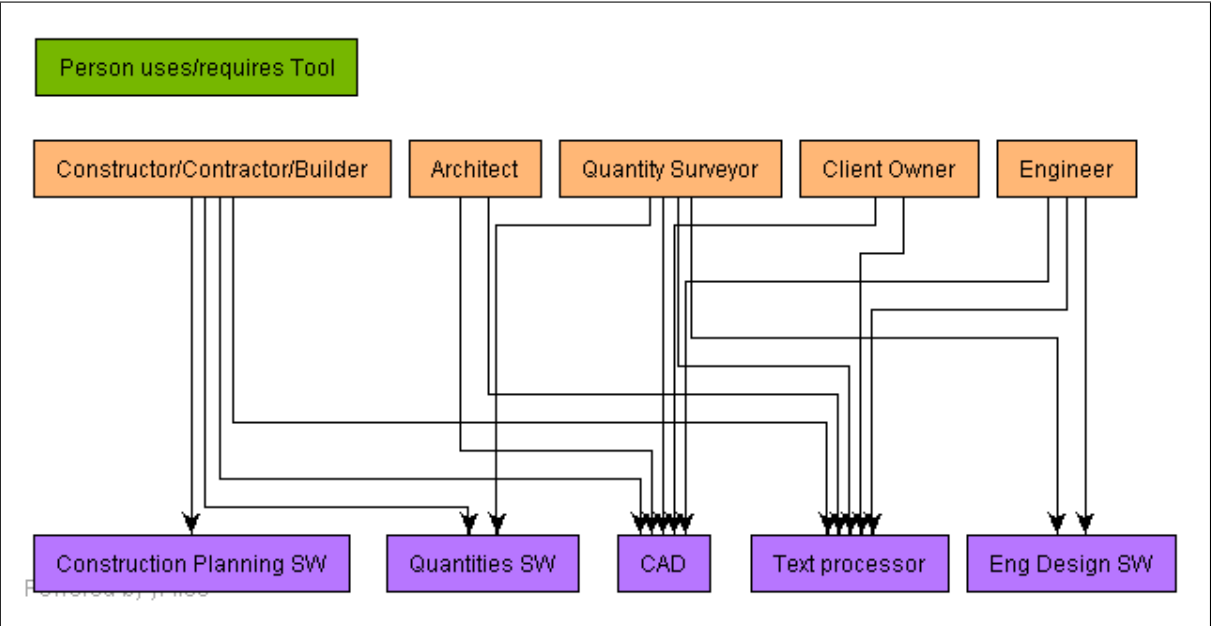


Figure 11.23: *Person uses tool*

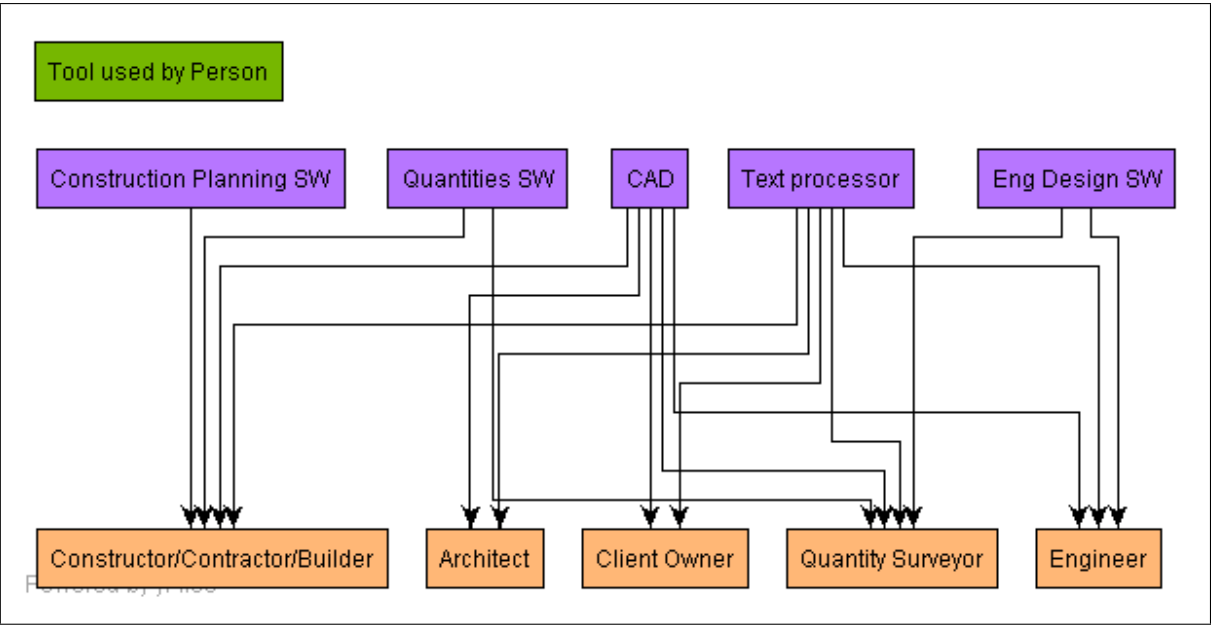


Figure 11.24: *Tool used by person*

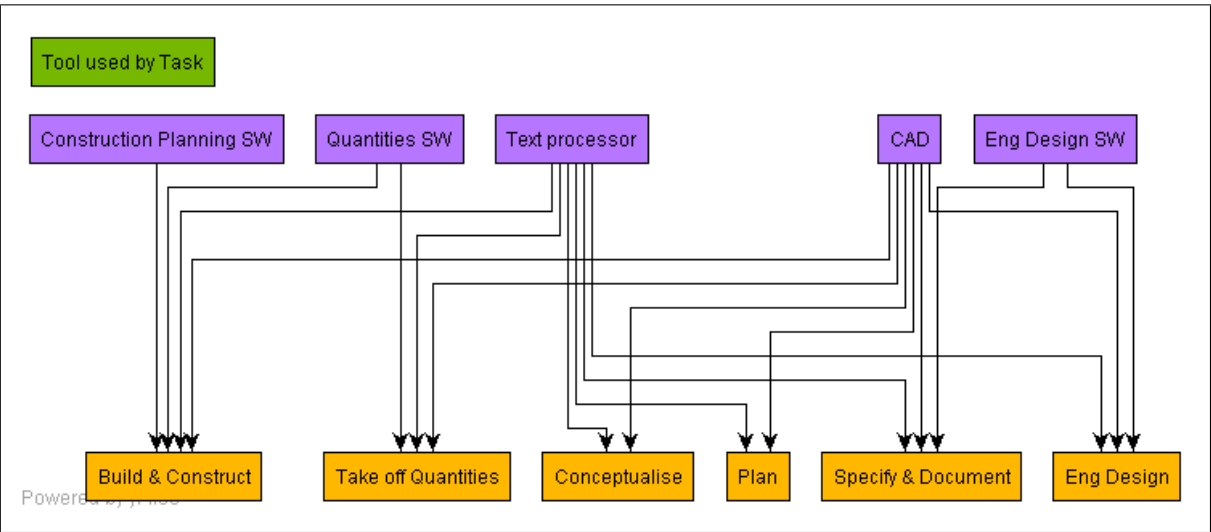


Figure 11.25: Data operated on by person

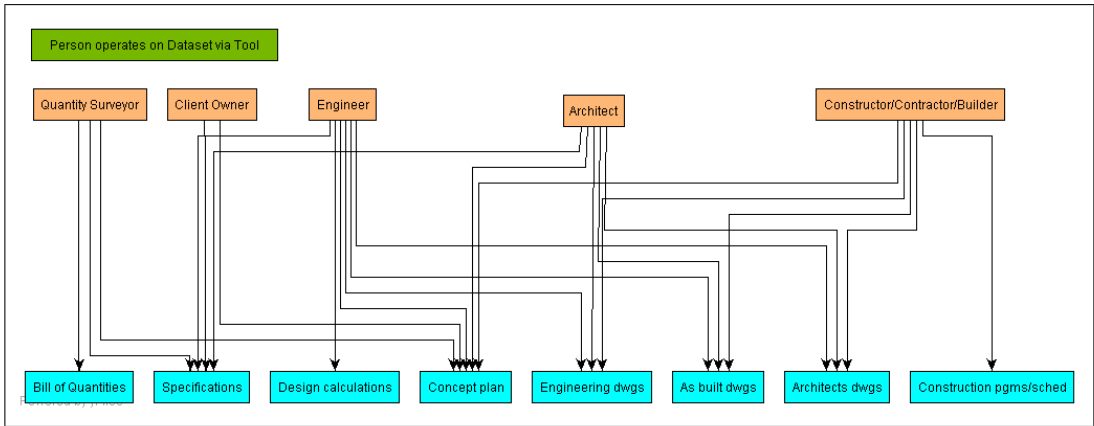


Figure 11.26: Person operates on dataset via tool

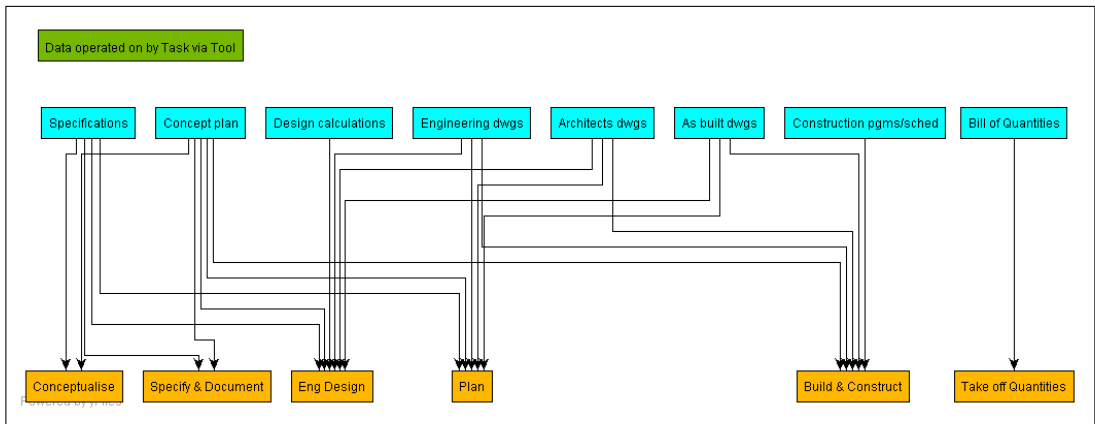


Figure 11.27: Data operated on by task via tool

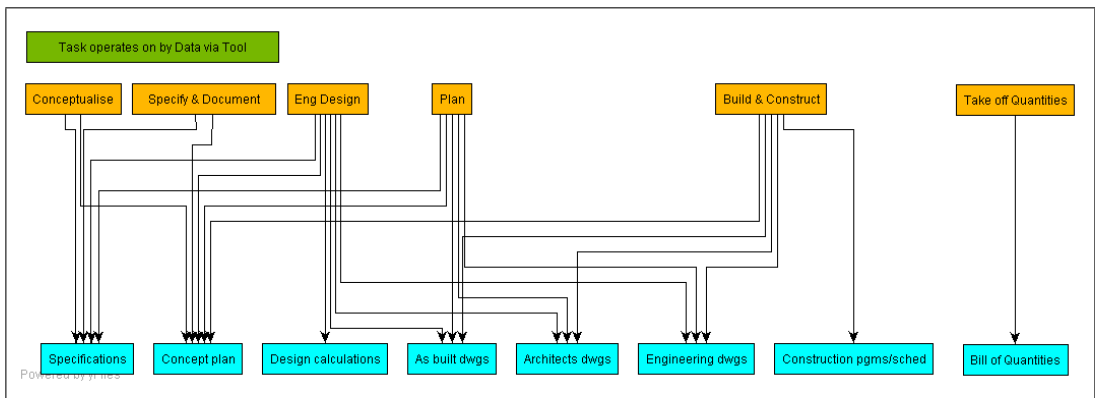


Figure 11.28: Task operates on dataset via tool

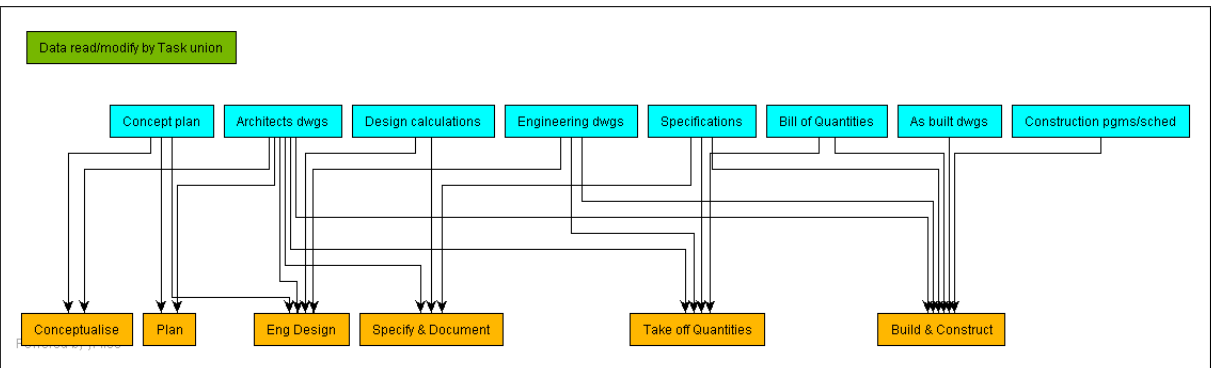


Figure 11.29: Data read modify via task union

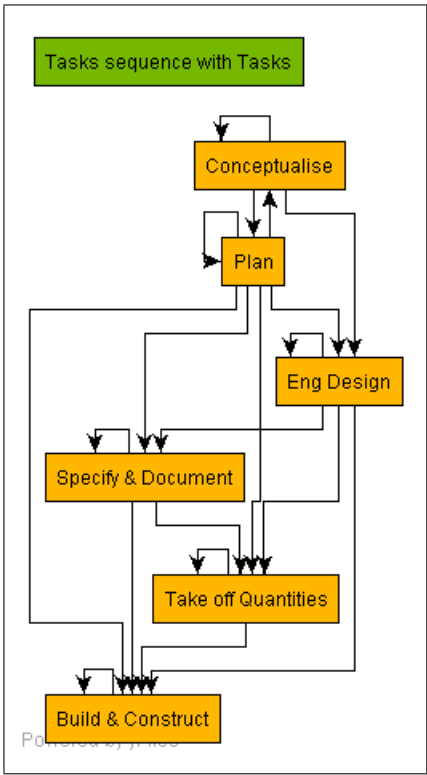


Figure 11.30: Sequence of tasks

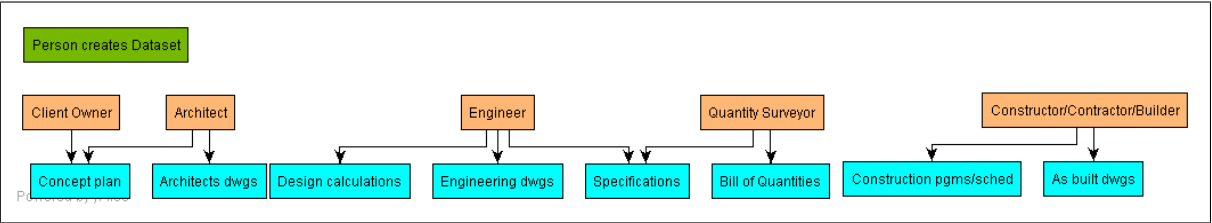


Figure 11.31: Person creates dataset

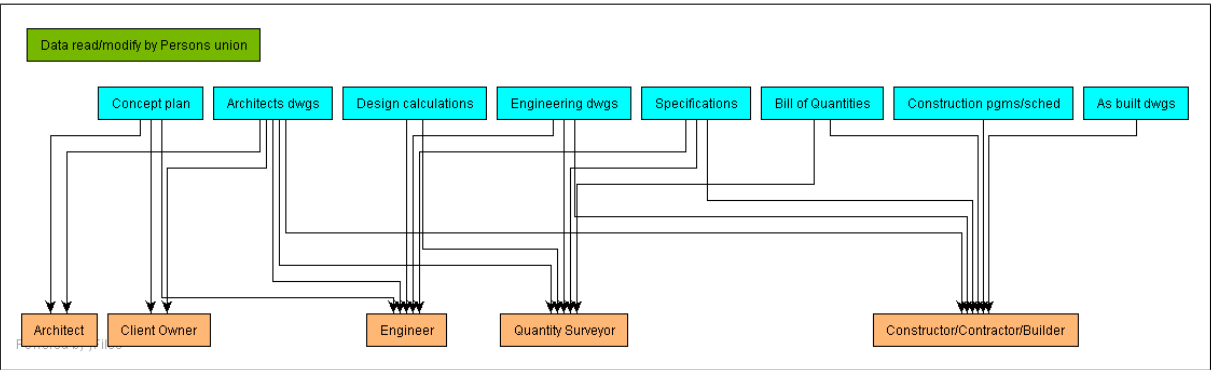


Figure 11.32: Data read modify via person union

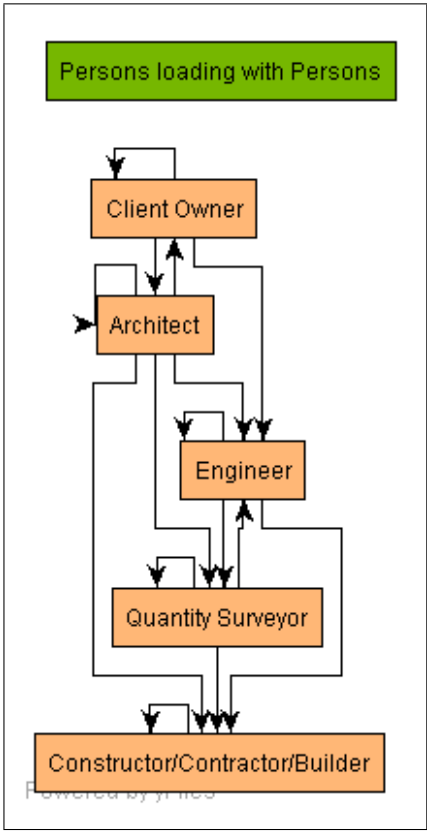


Figure 11.33: Person loading

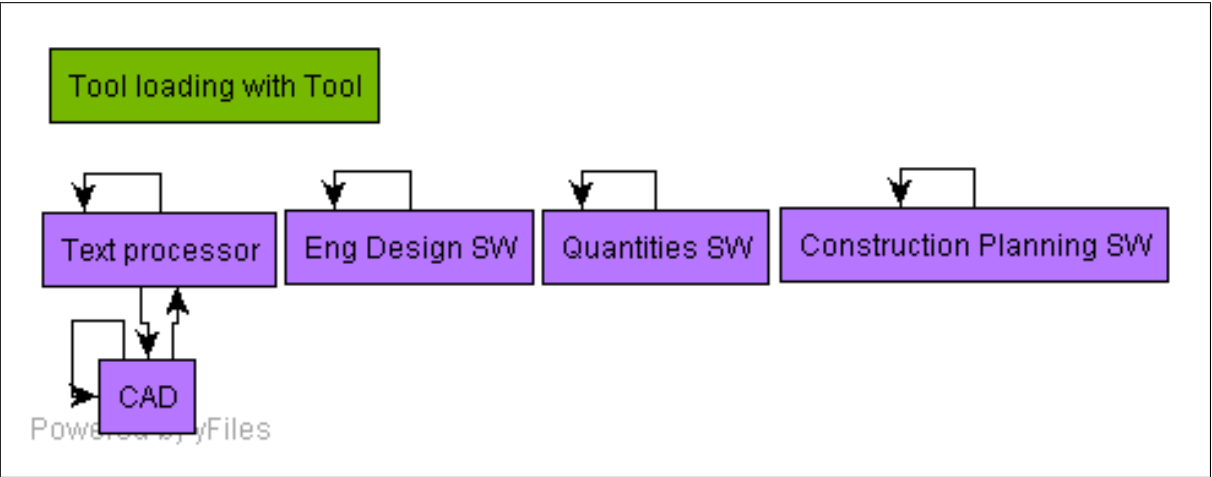


Figure 11.34: Tool loading - tools required with other tools

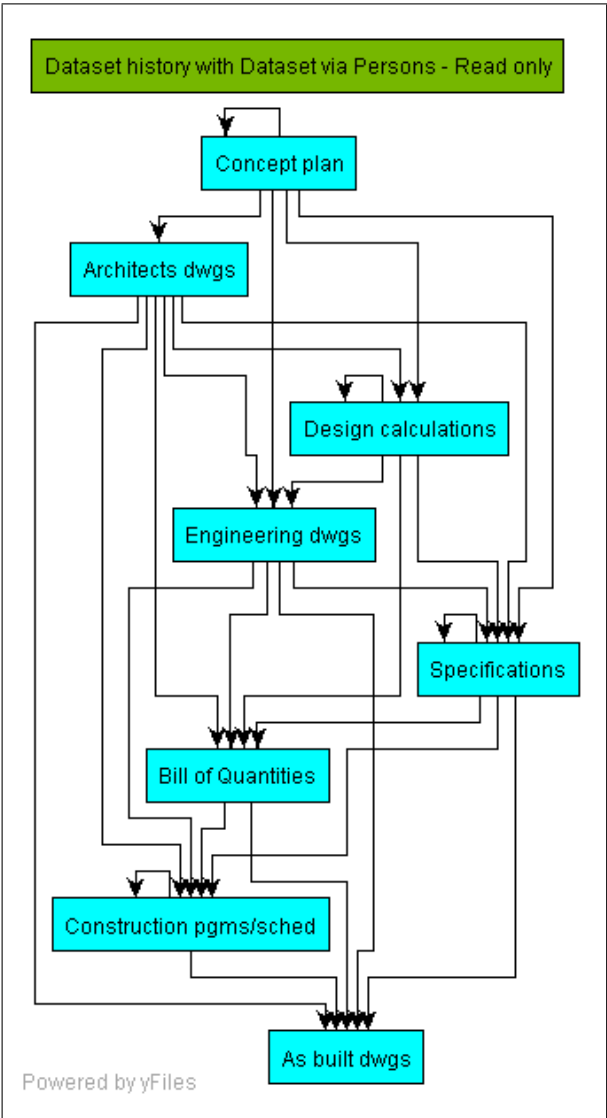


Figure 11.35: Dataset history with dataset determined via persons - read only

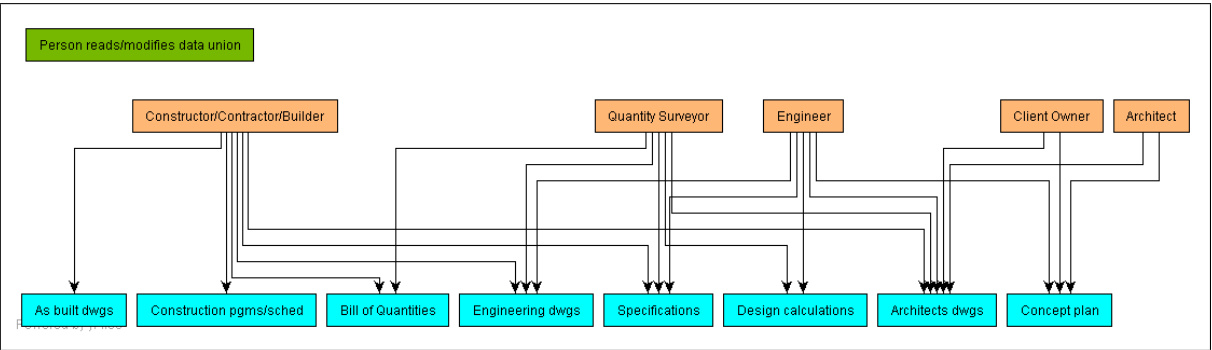


Figure 11.36: Person data read and modify via data union

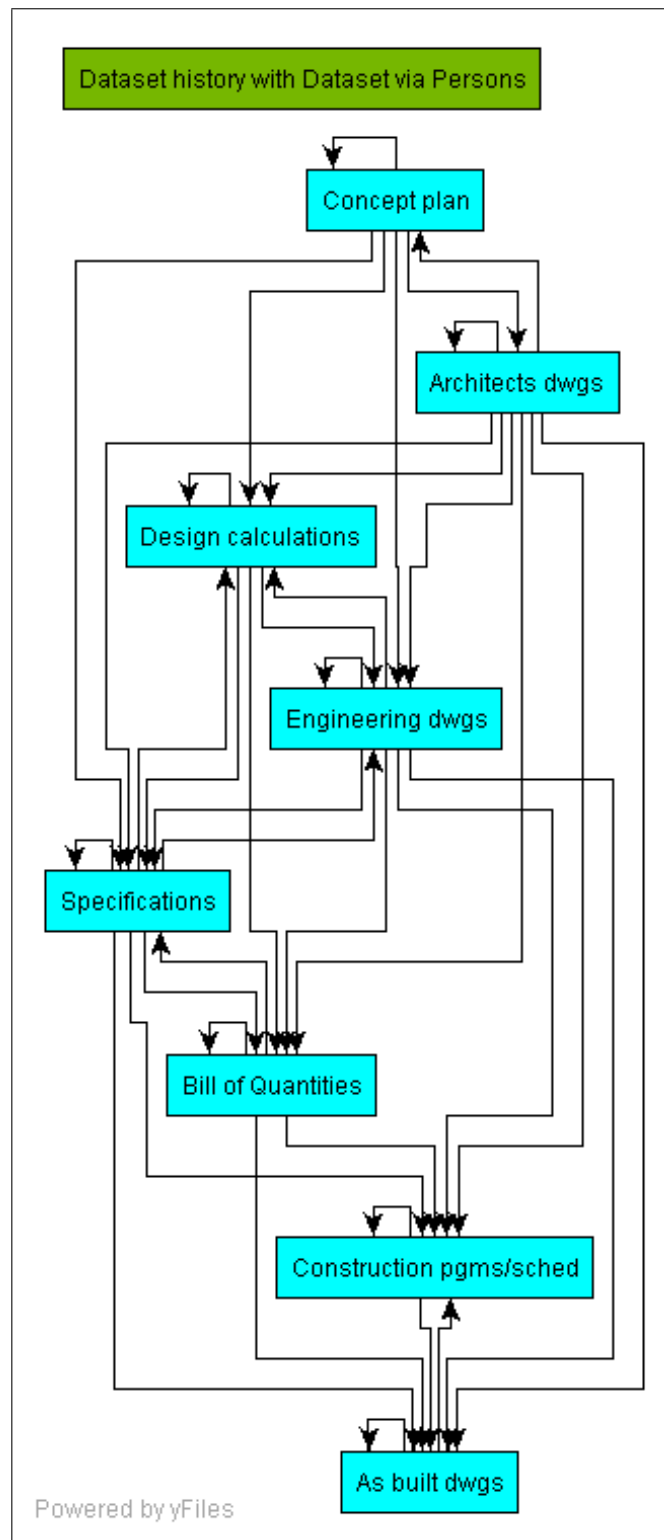


Figure 11.37: Dataset history with dataset determined via persons - read and modify

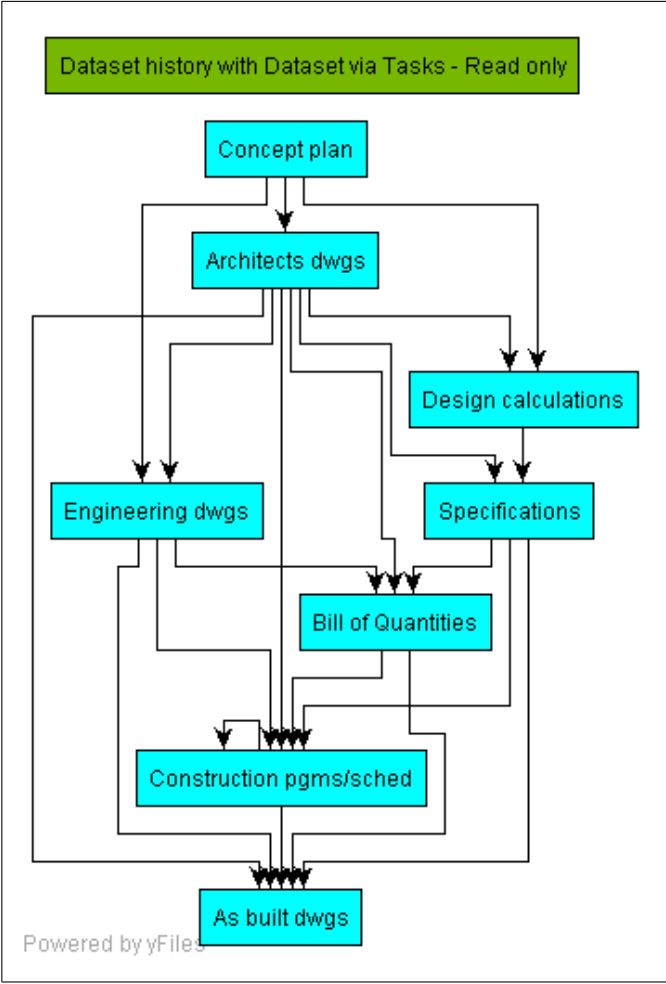


Figure 11.38: Dataset history with dataset determined via tasks - read only

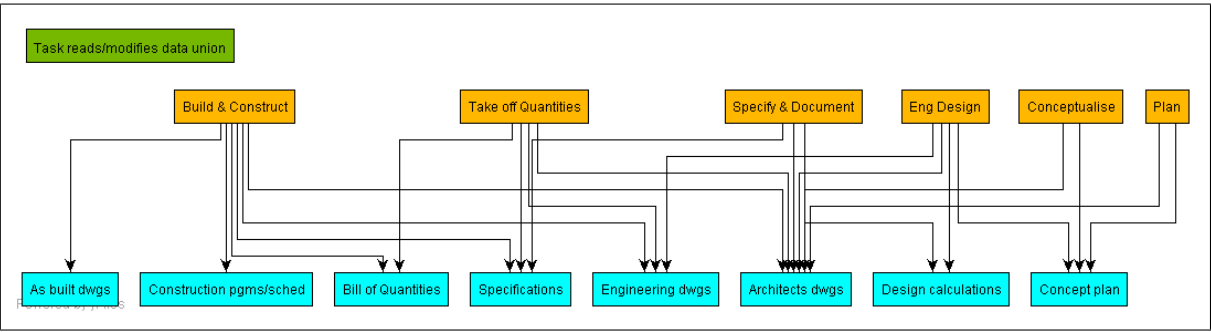


Figure 11.39: Task reads and modify via data union

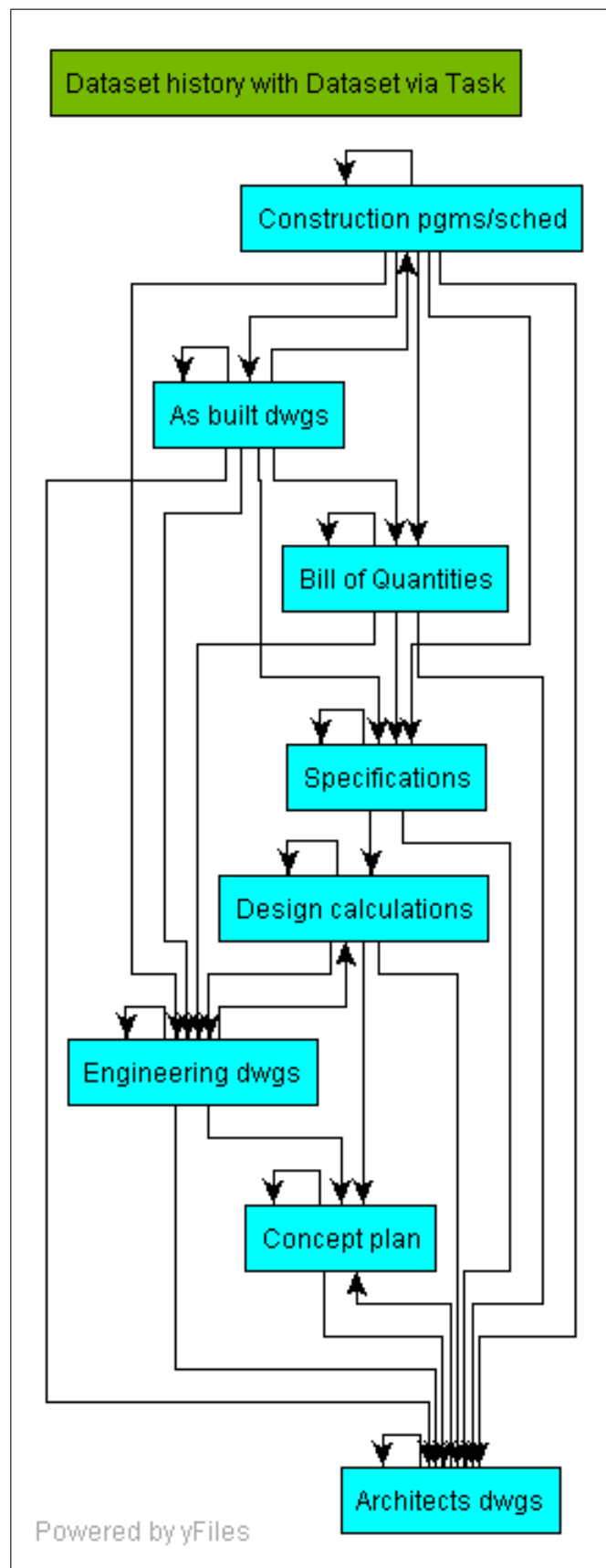


Figure 11.40: Dataset history with dataset determined via tasks - read and modify



Figure 11.41: Tasks and data for client - no tasks and data

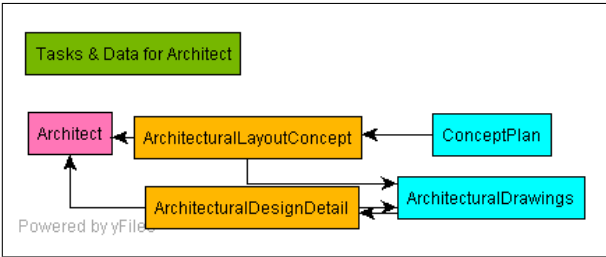


Figure 11.42: Tasks and data for architect

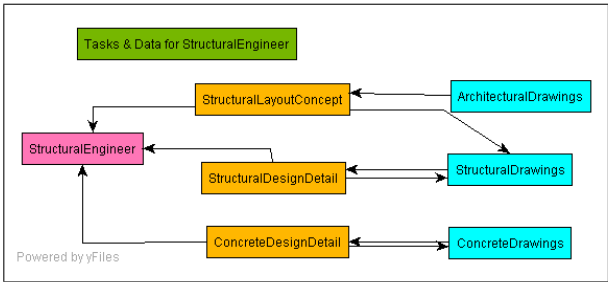


Figure 11.43: Tasks and data for structural engineer

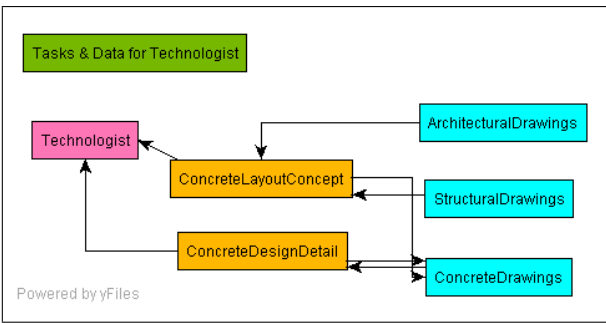


Figure 11.44: Tasks and data for technologist

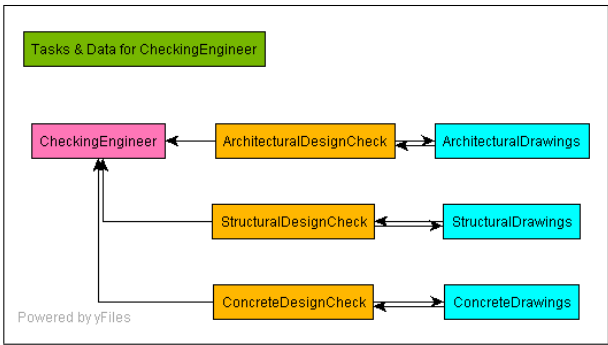


Figure 11.45: Tasks and data for checking engineer

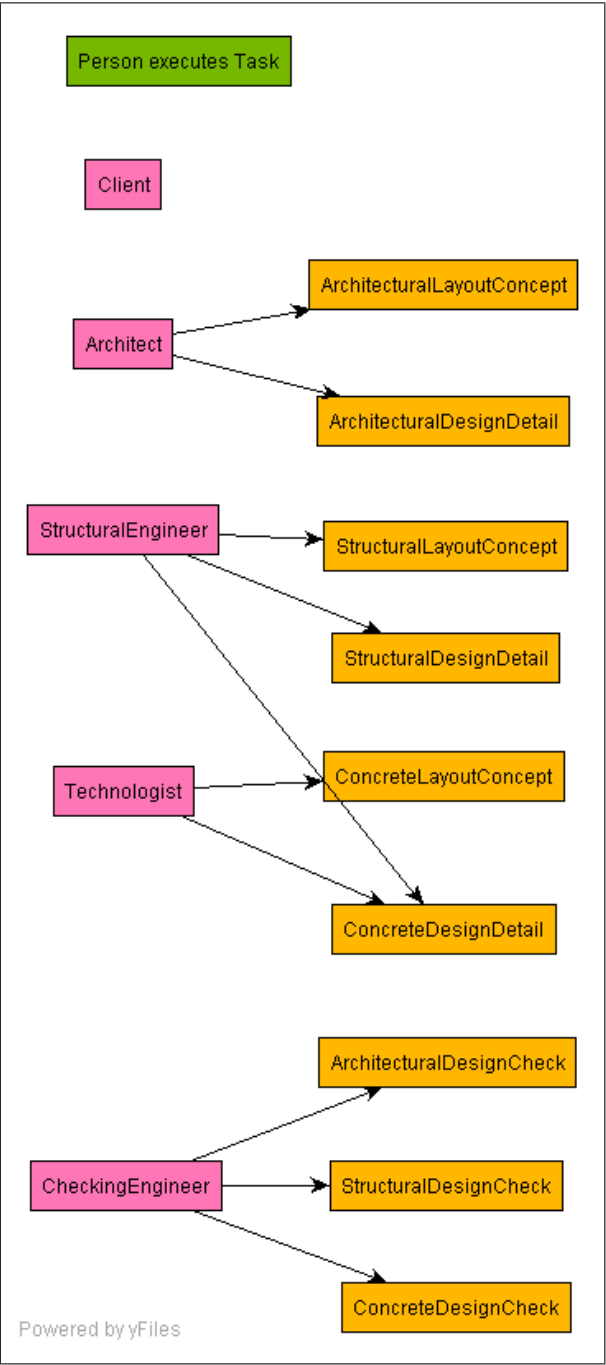


Figure 11.46: Person executes task relation

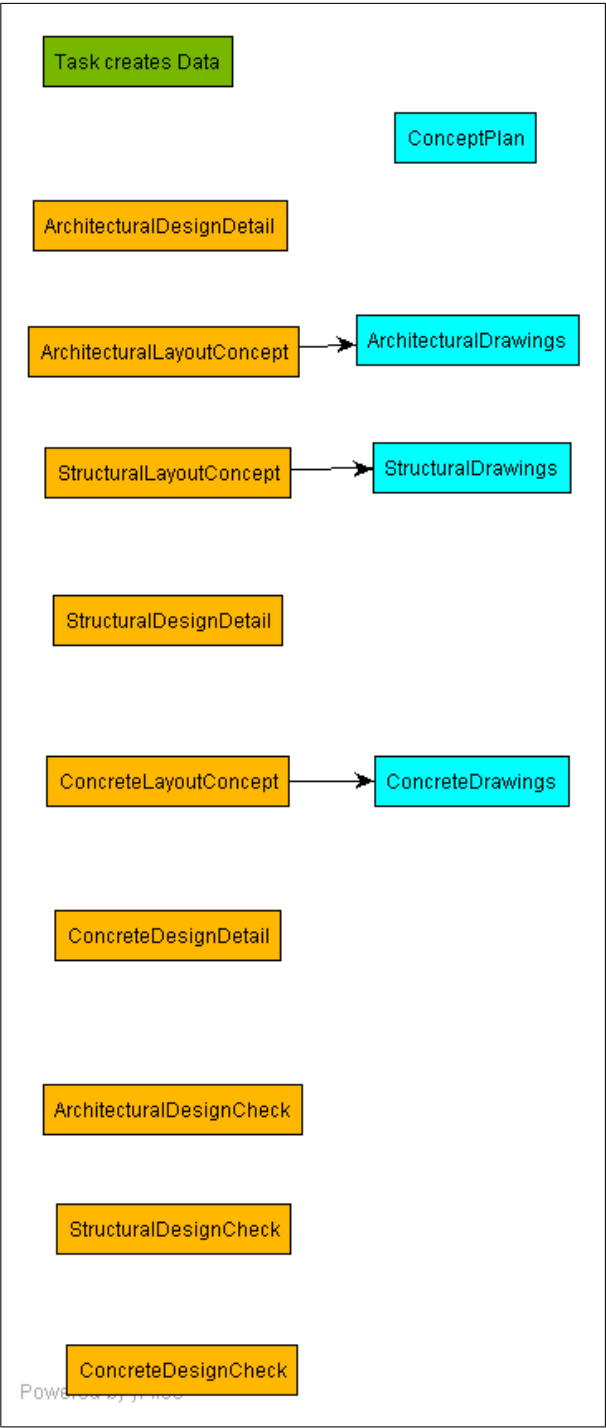


Figure 11.47: Tasks creates data relation

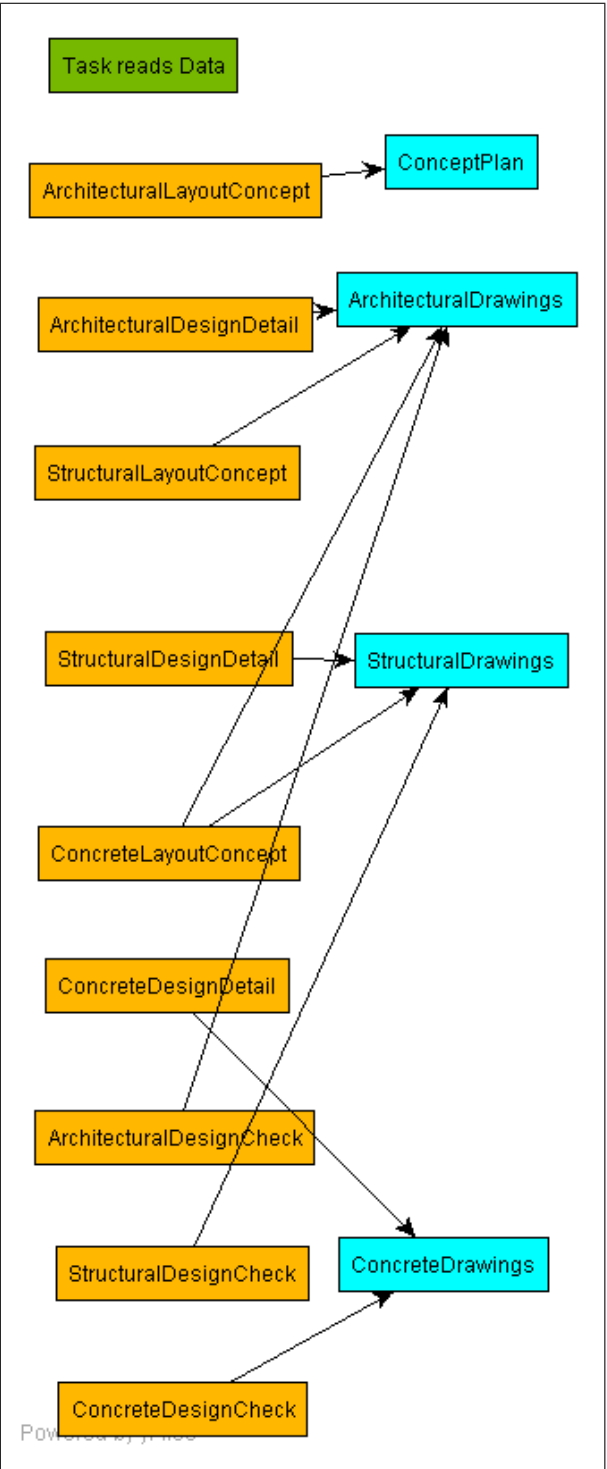


Figure 11.48: Tasks reads data relation

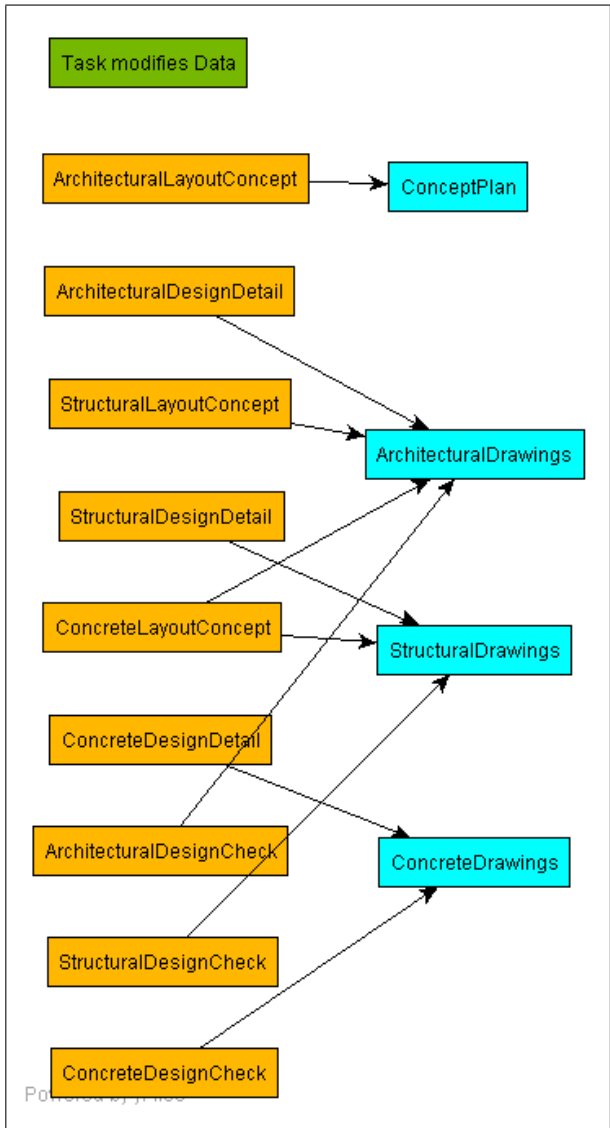


Figure 11.49: Tasks modifies data relation

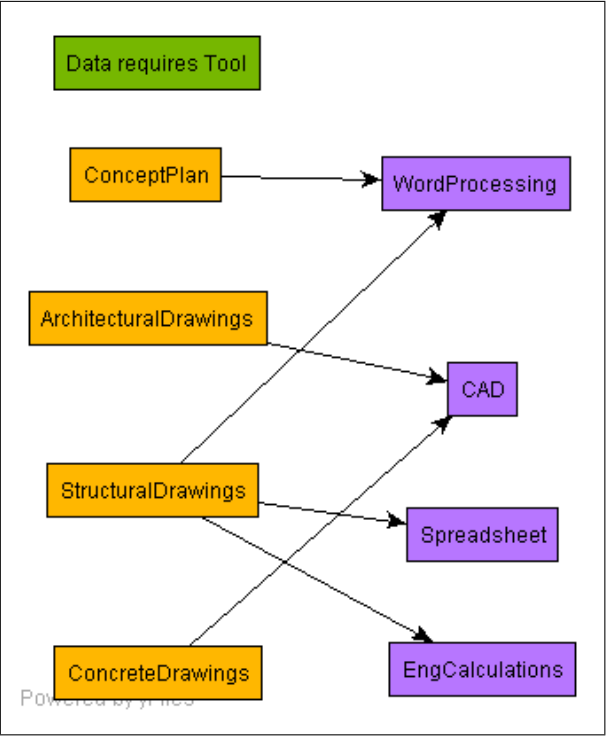


Figure 11.50: Data requires tool relation

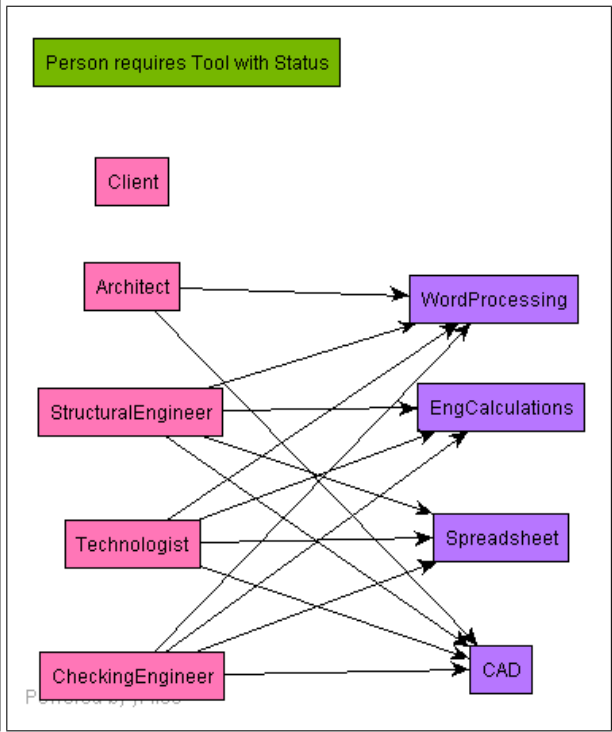


Figure 11.51: Person requires tool relation

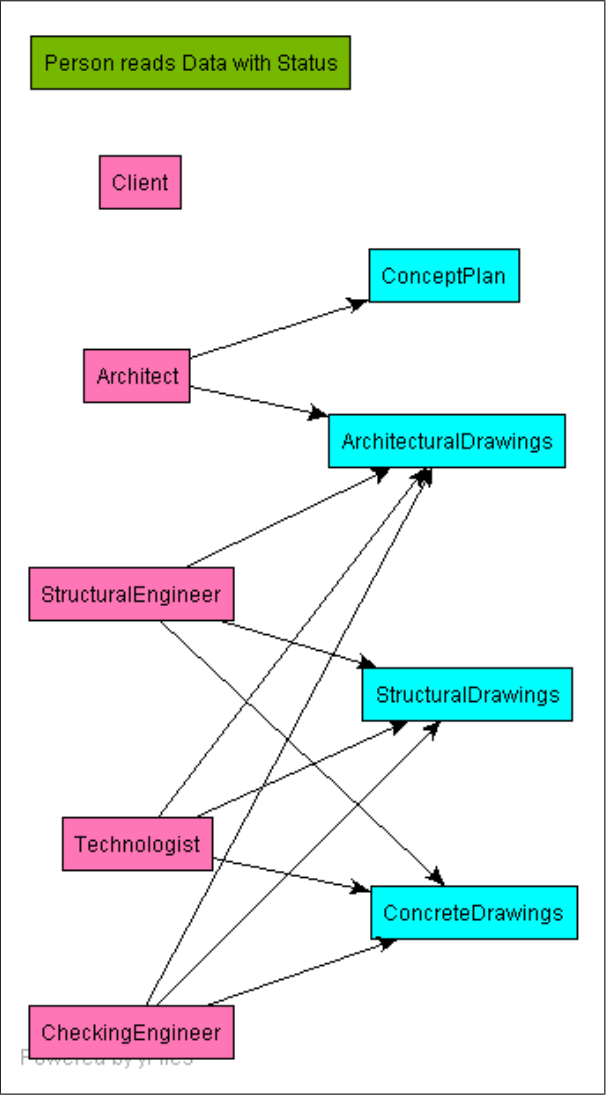


Figure 11.52: Person reads data relation

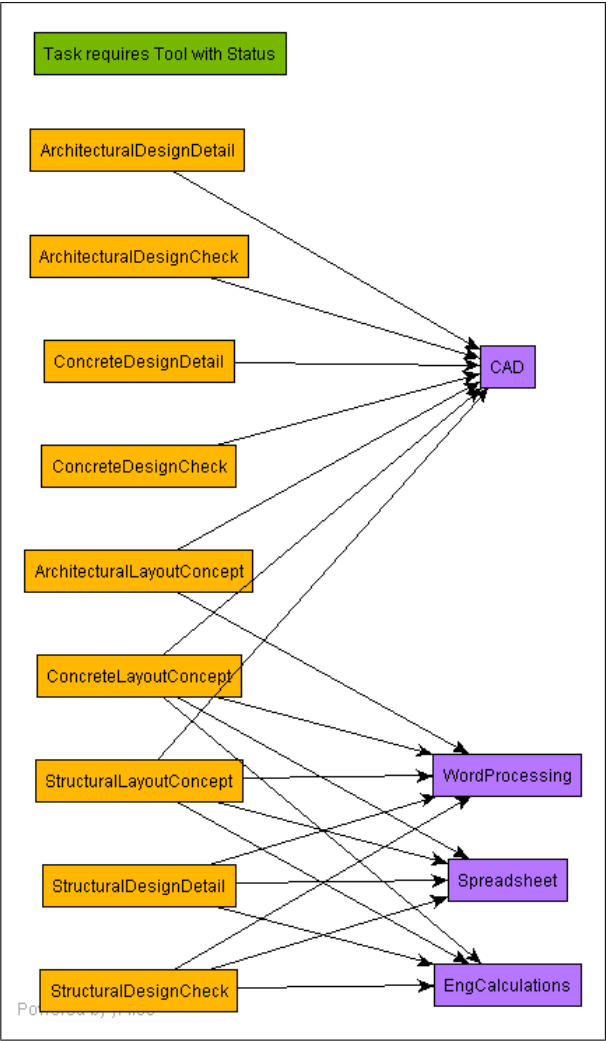


Figure 11.53: Tasks requires tool relation

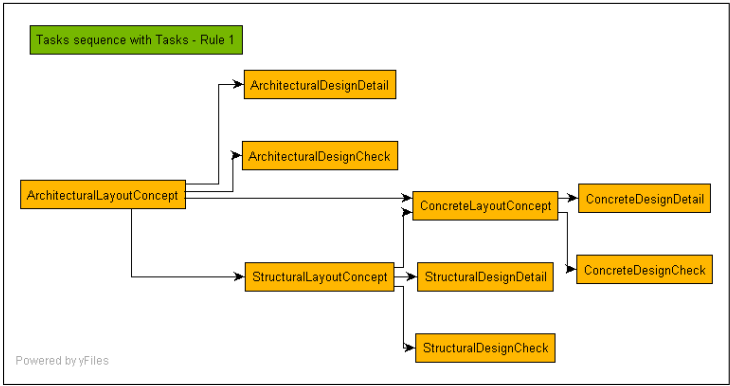


Figure 11.54: Task sequence - Rule 1

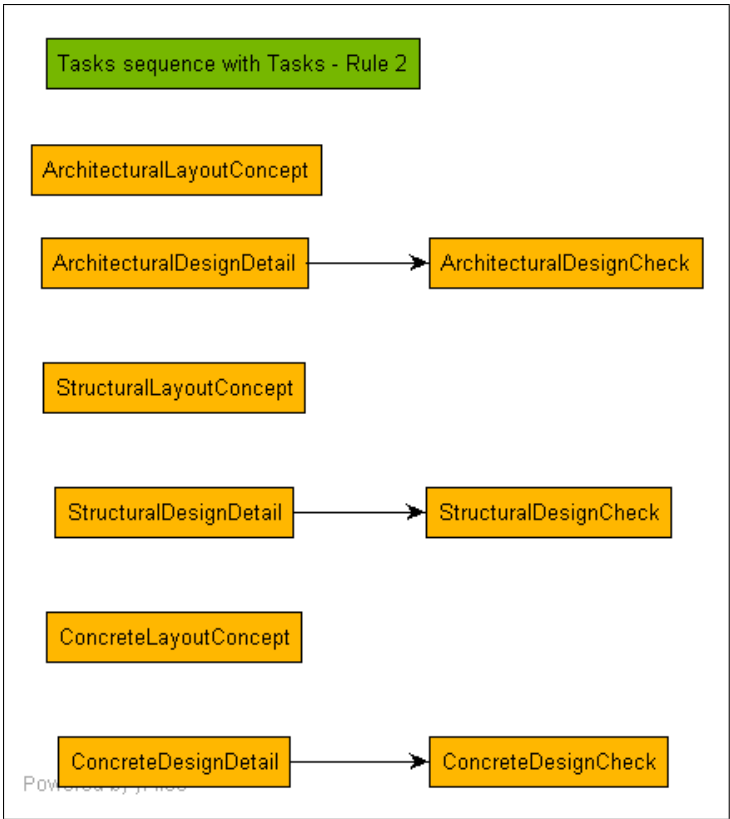


Figure 11.55: Task sequence - Rule 2

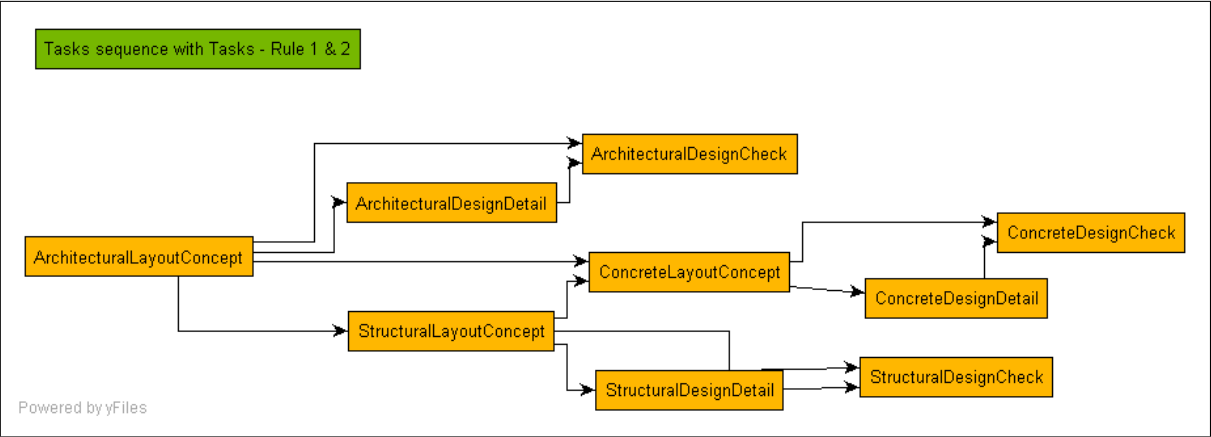


Figure 11.56: Task sequence - Rules 1 & 2

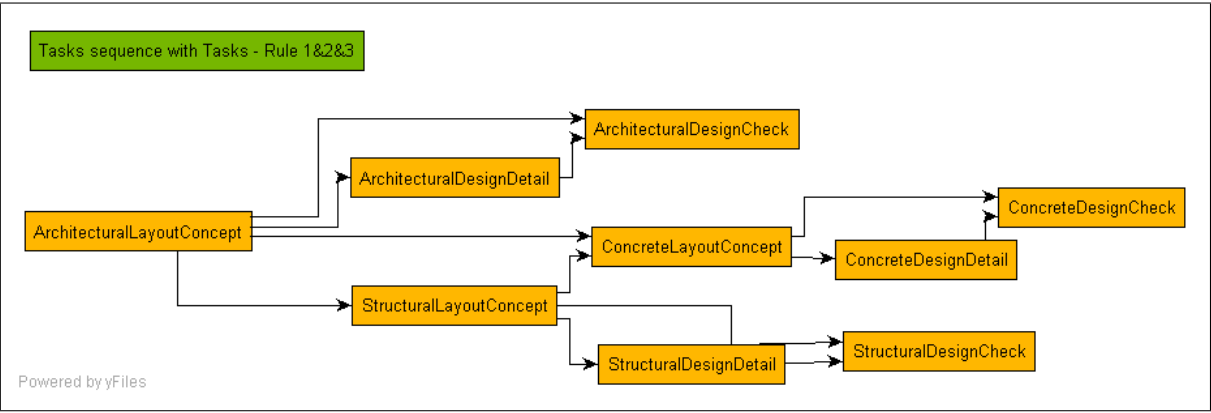


Figure 11.57: Task sequence - Rules 1, 2 & 3

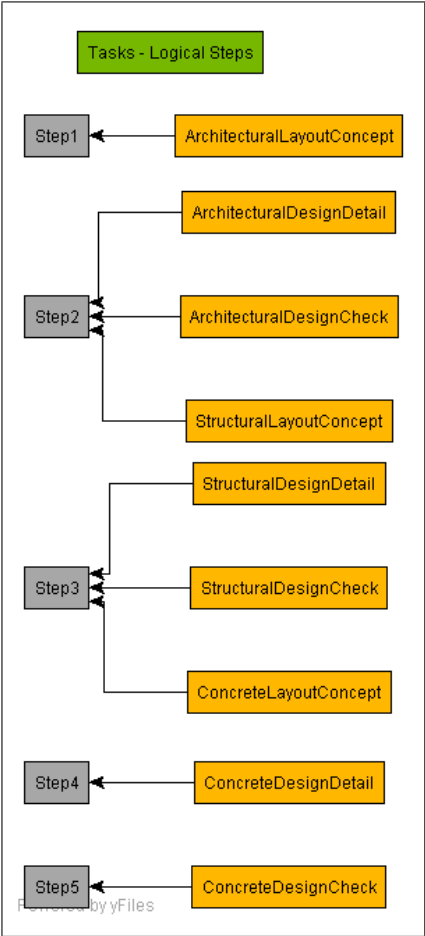


Figure 11.58: Step schedule of Tasks

Chapter 12

Representing and Processing management structure using graph applications

12.1 Introduction

The purpose of this section is to investigate the modelling of typical organisational management structures using a graph theory approach. The logical models of the structures can then be used to process management reporting data linked to the models.

Management and reporting structures and relations can be modelled using graphs. Once these graphs have been defined, business entities can be linked to the graphs and selected attributes of the entities in logical (boolean), numerical (e.g. monetary values) and literal (e.g. character string) data format can be collected and/or accumulated over the graph structure and a suitably defined report produced.

The graphs used are planar graphs and the forest is selected as the most general graph representing a structure in the analysis reported here. It should be possible to extend the approach to more general graphs and also make use of the path algebra approach to compute structured management report datasets. Refer to sections 5.6, 5.7, 5.7.2, 5.9.1, 5.9.4.2 and 5.10 for the definition of the graph theoretic concepts which apply here.

12.2 Typical Management Reporting Tree Structure

Consider the relation ' $person_x$ reports to $person_y$ ' in matrix format. On each level of management a relation can be defined describing the management hierarchy in tree format visualised as a plane graph.

Table 12.1 displays the boolean form of the relation of M_1 , M_2 , M_3 , M_4 reporting to M_A or M_B as indicated.

This adjacency matrix can be output in graphical format as shown in figure 12.1.

<i>ReportsTo</i>	M_A	M_B
M_1	1	
M_2	1	
M_3		1
M_4		1

Table 12.1: Management reporting relation

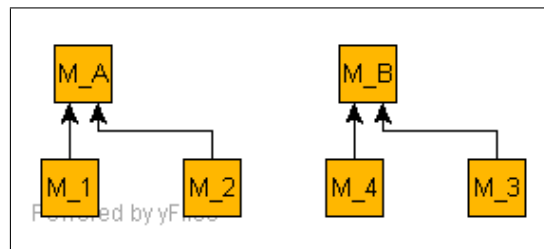


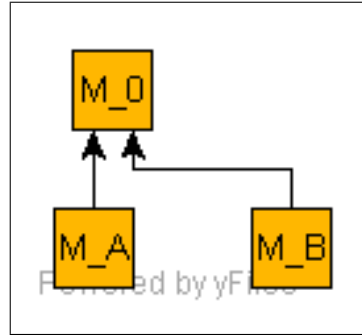
Figure 12.1: Reporting structure graph representation - level 1 - from adjacency matrix

<i>ReportsTo</i>	M_0
M_A	1
M_B	1

Table 12.2: Hihger level management reporting relation

On a next level a further relation can be defined between M_A and M_B reporting to M_0 as shown in table 12.2.

This adjacency matrix can be output in graphical format as shown in figure 12.2.

**Figure 12.2:** Reporting structure graph representation - level 2 - from adjacency matrix

The boolean representation in adjacency matrix format for the whole management structure is shown in table 12.3

This adjacency matrix can be output in graphical for as shown in figure 12.3.

12.3 Testing of the logic and integrity of the management structure

Testing of the logic and integrity of the management structure can be done by applying basic graph analysis. The list of items below contains an overview of a selection of operations which can be performed on the management structure graph representation to derive information on the underlying graph.

- (a) Number of managers active – Report the number of vertices in the graph – size of adjacency matrix
- (b) Number of relations between managers – Count the number of edges in the graph
- (c) Number of persons reporting to a manager and reported to per manager – refer to vertex indegrees
- (d) Lengths of paths in management structure adjacency matrix
- (e) Persons reporting to no manager at all (Top manager typically) – contained in a vertex outdegree analysis.
- (f) Testing if a relation between persons in the managerial tree exists – tracing paths

Given an adjacency matrix, one can check whether a given edge exists. To discover whether there is an edge, for each possible intermediate vertex v we can check whether (u, v) and (v, w) exist.

Table 12.3: Complete multi level management reporting relation

	M_0	M_A	M_B	M_1	M_2	M_3	M_4
M_0							
M_A	1						
M_B	1						
M_1		1					
M_2		1					
M_3			1				
M_4			1				

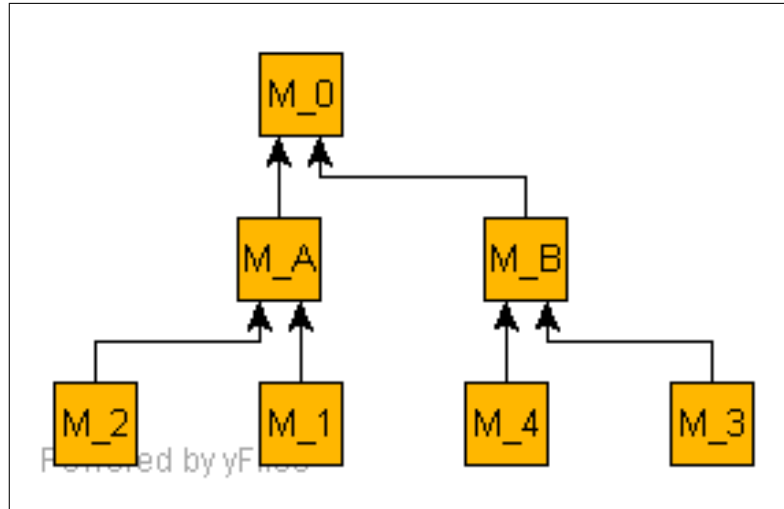


Figure 12.3: Reporting structure graph representation - all levels - from adjacency matrix

- (g) Test if the graph is a directed graph
- (h) Graph acyclicity can be checked
- (i) Checking for an isomorphism
- (j) Testing if the graph is a rooted tree or a forest
- (k) Lengths of paths in management structure adjacency matrix can be computed and reported

12.4 Converting from adjacency matrix format to adjacency list format

To represent a relation for computational purposes the boolean adjacency matrix representation, which is more appropriate for dense graphs, or the incidence list representation, which is more appropriate for sparse graphs, can be used.

The MATLAB code for the conversion from adjacency matrix format to adjacency list format is listed in appendix K, section K.6.1.

The MATLAB code for converting from adjacency list format to adjacency matrix format is also listed in appendix K, section K.6.2.

12.5 Depth first search and tree structure for a given graph

The depth first search algorithm (DFS) can be used to determine the spanning tree for a given graph given in adjacency matrix format. Once the spanning tree has been determined, subgraphs of the structure can be extracted to visualise the management reporting structure per level of management.

The spanning tree is also output in adjacency matrix format.

To determine the DFS tree the transpose of the adjacency matrix is input. DFS does not produce results on untransposed adjacency matrix.

The depth first search algorithm used is contained in the MATLAB shown in appendix section K.3.

Global variables are required due to the recursive nature of the algorithm. A double *while loop* is required in the code to ensure that unconnected graphs can be processed as well. All vertices need to be investigated before the search terminates.

The MATLAB function `DepthFirstSearch.m` uses the `search` function to find the next vertex linked to a given vertex and returns 0 if no vertex can be found.

The resulting trees have adjacency matrices which are the transpose of the original structure adjacency matrices as shown in figure 12.4.

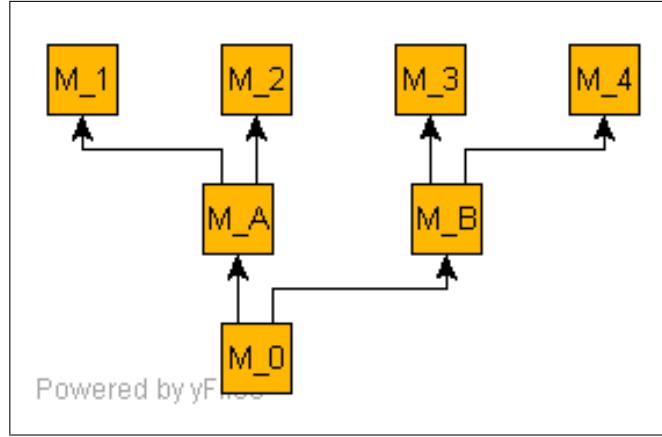


Figure 12.4: *Reporting structure adjacency matrix*

12.6 Inserting sub management structures into larger structures

Adjacency matrices containing representations of management structures can be combined by a submatrix insertion process to yield higher level complete representations in adjacency matrix format. Vertex labelling of input graphs referred to vertex labels of combined graphs are used.

The MATLAB code listed in appendix section K.5 builds up the vertex label sets and the total graph representation.

12.7 Extracting subgraphs of vertices linked to a selected vertex

Algorithms were developed to extract subgraphs with the vertices linked to a selected vertex of a given graph.

The MATLAB algorithm shown in appendix section K.6 traverses the adjacency list and enters the edges linked to a selected vertex into a adjacency matrix which has the size of the original graph.

As vertices on active edges are encountered they are marked as being active in the subgraph in a list of vertices to be retained as used by the subgraph. In the final step of the algorithm only adjacency matrix rows and columns of active vertices are collected in the adjacency matrix of the subgraph extracted.

12.8 Management/ reporting structure - tree analysis examples

12.8.1 Basic example

The three level reporting structure used previously serves as the starting point for this example. Refer to figure 12.5.

The adjacency matrix in this case is shown in equation 12.8.1.

$$\mathbf{T}_{in} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (12.8.1)$$

Applying the topological sorting to the transpose of the adjacency matrix yields the assignment of vertices to the structural levels of the hierarchy shown in matrix adjacency format in equation 12.8.2.

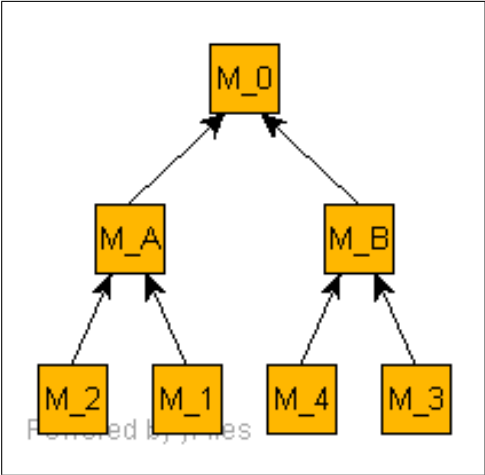


Figure 12.5: Reporting structure adjacency matrix

$$\mathbf{T}_{out} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \quad (12.8.2)$$

The plot of the graph in this case with vertices L_0 , L_1 and L_2 assigned to the column vertex references is shown in figure 12.6. L_0 , L_1 and L_2 represent the levels which each vertex in the graph is assigned to as shown in table 12.4.

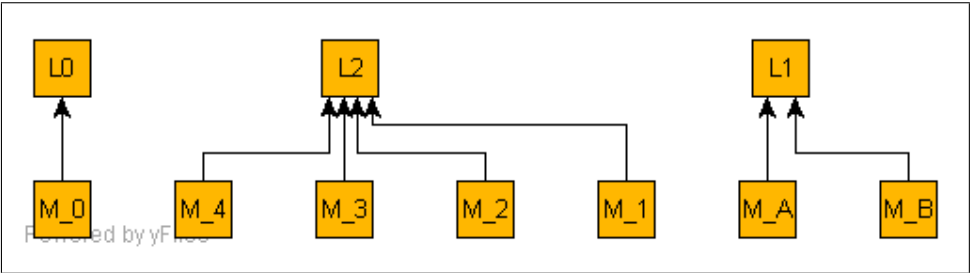


Figure 12.6: Reporting structure adjacency matrix

Applying the algorithm leads to the two submatrices displayed graphical format in figure 12.7. The graphical representation of these adjacency matrices are shown in figure 12.8. The matrices computed above can be used to process the attributes assigned to lower levels in a ‘roll up’ process for management reporting at a higher level.

Table 12.4: Management reporting graph levels

Level	Vertices in level of tree graph
L_0	M_0
L_1	M_A, M_B
L_2	M_1, M_2, M_3, M_4

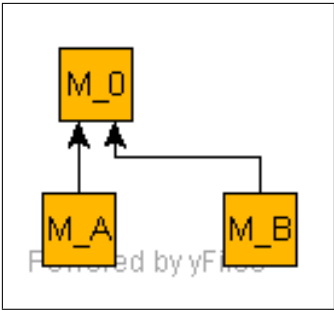


Figure 12.7: Reporting structure adjacency matrix - Level 1 to Level 0

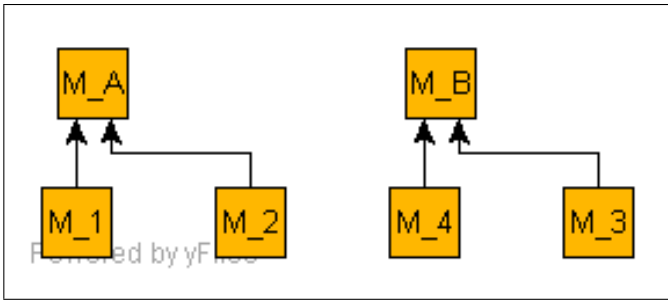


Figure 12.8: Reporting structure adjacency matrix - Level2 to Level 1

12.8.2 Larger more realistic example

A computed task step schedule shown in figure 12.9 is used to demonstrate the extraction of the subgraphs to determine the tasks which follow on any given task. Note that a header vertex is added to level L_0 of each graph in this case to be used for identifying the diagram. Table 12.5 lists the tasks contained in the schedule.

The topological sorting of the tasks in graph format is shown in figure 12.10.

Figures 12.11, 12.12, 12.13, 12.14 and 12.15 show the sub-adjacency graphs for the tasks of each logical step.

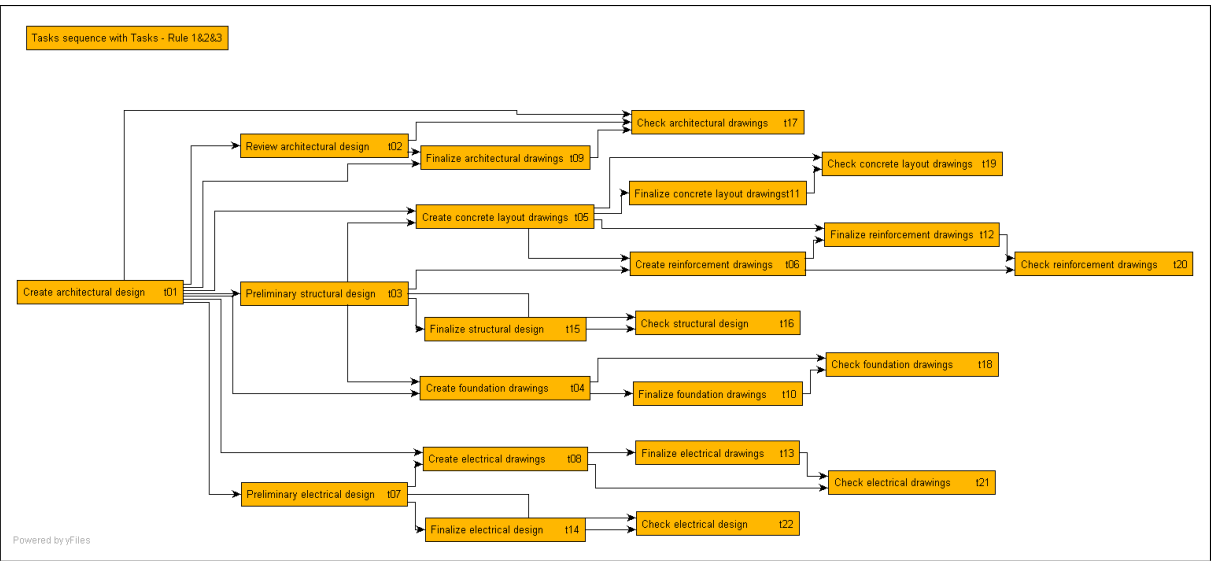


Figure 12.9: Task step schedule adjacency matrix

Table 12.5: Tasks for graph structure analysis

1	Create architectural design t01
2	Review architectural design t02
3	Preliminary structural design t03
4	Create foundation drawings t04
5	Create concrete layout drawings t05
6	Create reinforcement drawings t06
7	Preliminary electrical design t07
8	Create electrical drawings t08
9	Finalise architectural drawings t09
10	Finalise foundation drawings t10
11	Finalise concrete layout drawings t11
12	Finalise reinforcement drawings t12
13	Finalise electrical drawings t13
14	Finalise electrical design t14
15	Finalise structural design t15
16	Check structural design t16
17	Check architectural drawings t17
18	Check foundation drawings t18
19	Check concrete layout drawings t19
20	Check reinforcement drawings t20
21	Check electrical drawings t21
22	Check electrical design t22

Refer to appendix section K.7.2 for a MATLAB listing of the sub-adjacency graphs with abbreviated adjacency matrices for this example.

12.9 Determining connectivity of vertices in graphs to determine tree vertex links for roll up of reports

An extension of the logic described in section 12.7 is used to extract the sub tree of a graph linked to any given vertex. This data is required to drive the data roll up process described in section 12.10 below.

The logic for extracting the connectivity of a given vertex in a graph is given in the MATLAB code. Edges from vertices leading into a given vertex is determined and the subgraph extracted.

Refer to appendix section K.11 for an example application. The MATLAB functions developed are listed in appendix section K.8.

12.10 Report data roll up using adjacency matrices

Two alternative approaches to report data roll is outlined in this section.

The approaches use:

- topological sorting of the graph with subgraph extraction
- adjacency list processing

Examples with numerical values linked to graph vertices as well as text strings linked to graph vertices are given. The reader is referred to appendix K.

The progression of the data summing - or string concatenation process, driven by the graph structure, can be seen in the listings of the MATLAB programs.

Theoretical Example using topological sorting and subgraph extraction

Refer to section K.9.1 for an example application.

Theoretical Example - Using adjacency list processing.

Refer to section K.9.2 for an example application.

Larger example with numerical values

Refer to section K.9.3 for an example application.

Larger example with string literal values concatenated in accumulation process

Refer to section K.9.4 for an example application.

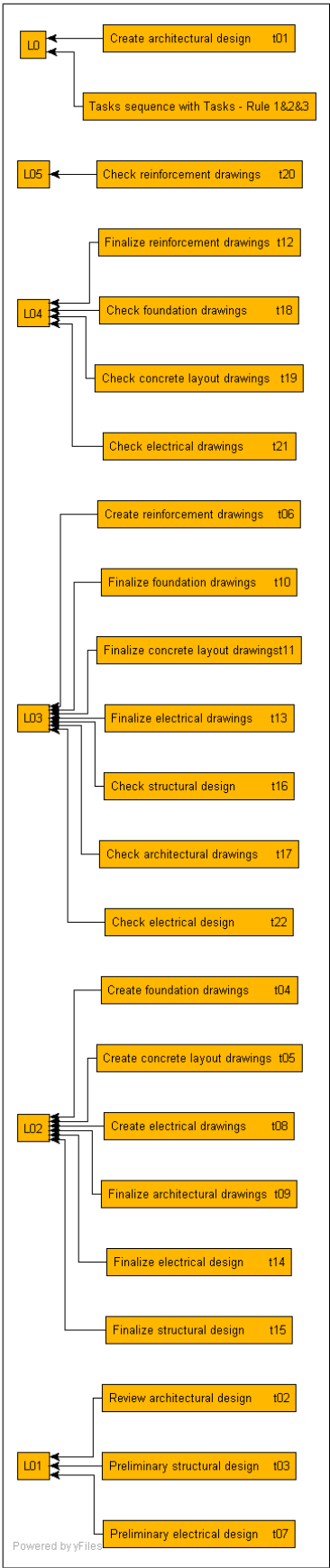


Figure 12.10: Topological sorting of the task step schedule

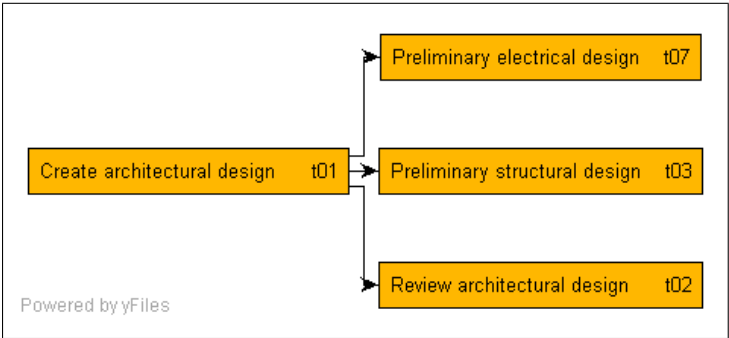


Figure 12.11: Sub-adjacency matrix for logical step 1

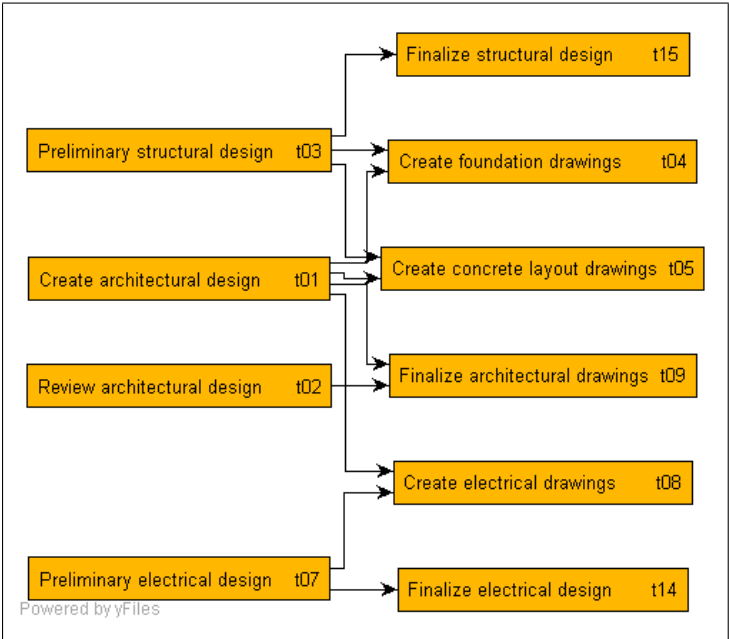


Figure 12.12: Sub-adjacency matrix for logical step 2

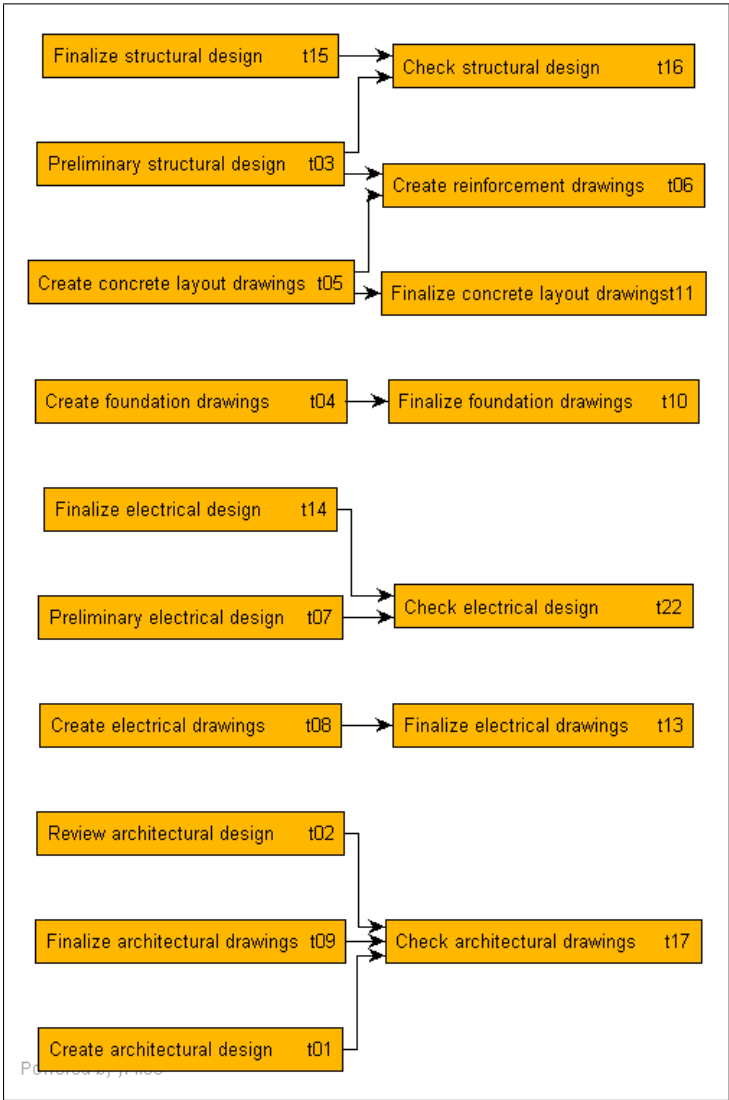


Figure 12.13: Sub-adjacency matrix for logical step 3

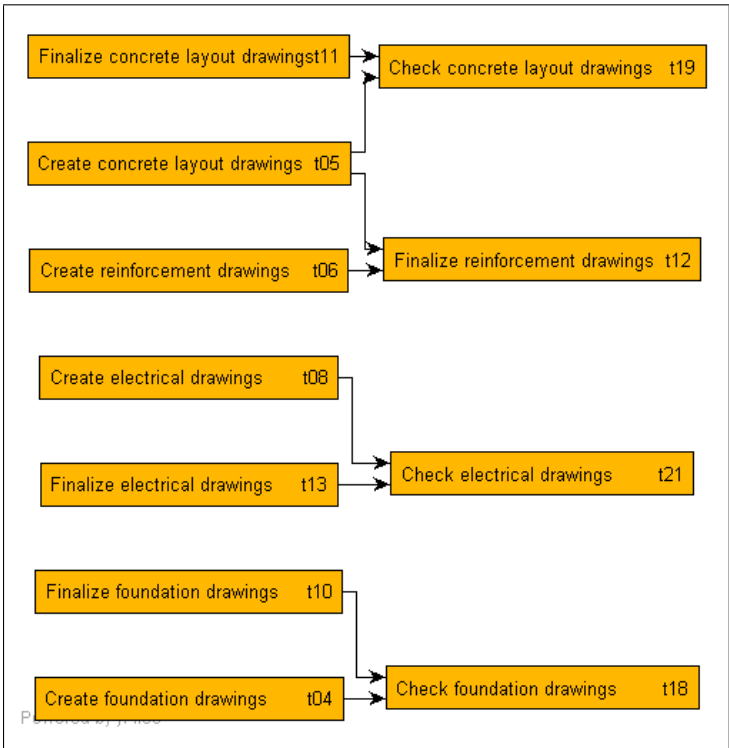


Figure 12.14: Sub-adjacency matrix for logical step 4

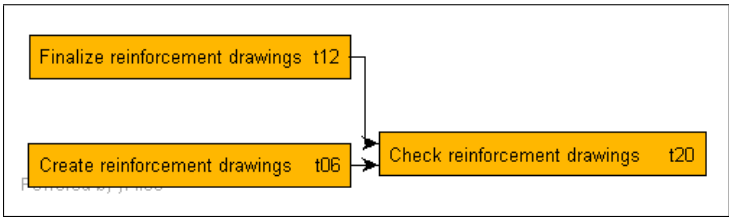


Figure 12.15: Sub-adjacency matrix for logical step 5

Chapter 13

Engineering Process Model: Database development, processing and report generation

13.1 Introduction

This chapter deals with the development of a database version of the Engineering Process Model. The motivation is to demonstrate the application of the techniques of the model using typical commercial and open source:

- hardware
- operating systems
- database applications
- database development tools

Detail data examples, source code samples and typical system and program output are contained in appendix J.

13.2 Database demonstration system overview

A basic Unified Modelling Language (UML) deployment diagram of the database development and demonstration environment is shown in figure 13.1. The deployment is chosen to represent a typical one which can be used in the services business enterprise. It makes use of proprietary and open source operating systems and application programs.

Information on the Unified Modelling Language applied to systems can be found in Booch et al. [18], Naiburg and Maksimchuk [85], Alhir [3], Bennett et al. [15] and Gomaa [43].

13.2.1 Client-server configuration

The user (client) uses a desktop or notebook computer which uses the Microsoft Access Database application program and the Microsoft Windows XP operating system to access the sever. The Microsoft Windows XP operating system is implemented on Intel processor based hardware.

The client-server environment server application, is set up on an Intel processor based hardware using the Red Hat Distribution Linux operating system (Red Hat Inc. [100]) server with the PostgreSQL (The PostgreSQL Global Development Group [121]) database management software loaded.

Interaction with the server for database maintenance and programming is done via the SSH Secure Shell Client terminal (SSH Communications Security [114]) to the Linux Bash shell as well as the PSQL database terminal program for the PostgreSQL database management software.

13.2.2 Network configuration

The network link between the client and server systems was based on the Microsoft ODBC database connectivity functionality over a TCP/IP link using port 5432. The physical layer of the network was over an Ethernet/SDSL/Ethernet link to a remote location.

Database demonstration system deployment

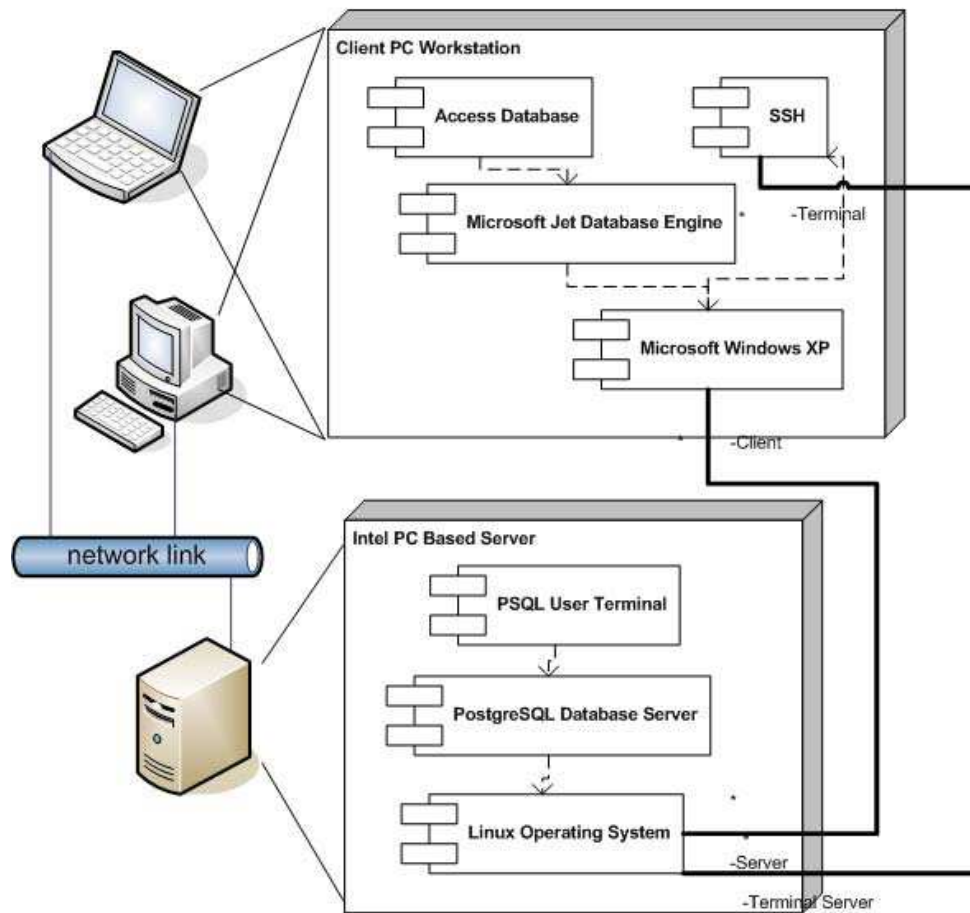


Figure 13.1: Database demonstration UML Deployment Diagram

13.3 Defining the database structure

The Azzurri (Azzurri Limited [10]) Clay Eclipse plug-in database modelling program was used to develop the database schema for the Engineering Process Model. Refer to The Eclipse Foundation [119] for more information on the Eclipse development environment.

According to the Azzurri web site:

‘Clay is a database design tool that runs as a plug-in in the Eclipse development environment. Clay has an intuitive user interface for graphically designing database models. Clay can also create a database model by reverse engineering an existing database. Furthermore, Clay generates the SQL (DDL) code appropriate for your database.’

The tables were named using the ‘tb’ prefix naming convention. A first normal form structure was used for the database. Refer to section 7.17.

The process of defining the database using Clay includes the following:

- define table names
- define fields with field types
- select primary and other keys
- link fields to table keys as required

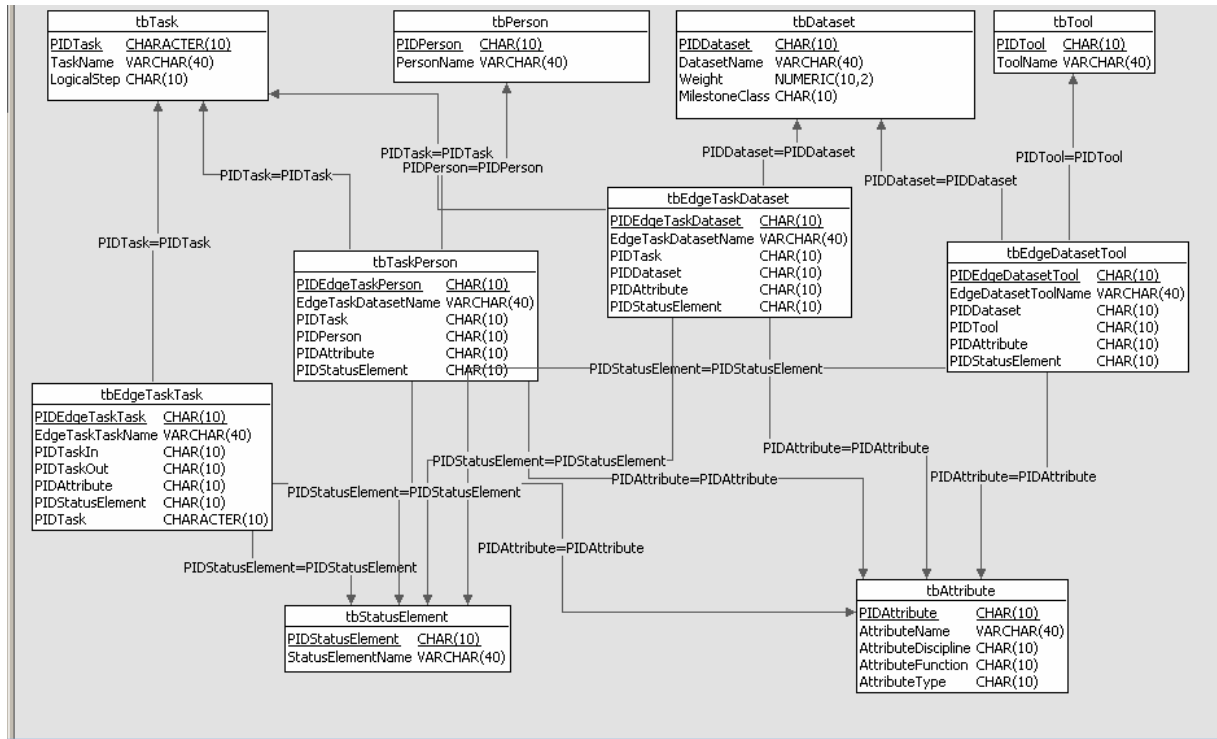


Figure 13.2: Engineering Process Model - Clay program screen

Figure 13.2 contains an overview of the database tables designed. Figure J.1 contains a more detailed overview of the database tables designed.

More information on the software used is given in:

- Appendix J.1 – Eclipse software – The Eclipse Foundation [119]
- Appendix J.2 – Azzurri Clay software – Azzurri Limited [10]

13.3.1 Azzurri Clay XML DTD Specification file

The Clay program stores the data describing the database design in a *.clay* file. This file is in *XML* format.

The structure of the *.clay XML* file is described in the *XML* Data Type Definition (DTD) specification given in appendix section J.2.

The well formedness of a *.clay* data file can be checked by a program such as XMLSpy by Altova (Altova, Inc. [4]) and the file validated using the *XML* file DTD specification listed in appendix section J.3.

13.3.2 Database setup SQL statements

Once the database design has been done the SQL data definition statements needed to build the database are generated by the Clay program and saved in a *.sql* file. A selection of SQL standards are available. The SQL statements were generated using the ANSI-92 Standard. The database definition file was transferred to the Linux server. The contents of a typical file is listed in Appendix J.4.

13.3.3 PostgreSQL Database Reference

The database definition file which was transferred to the Linux server was used to set up the database on the server. The functionality of the PostgreSQL server terminal, i.e. PSQL, was used to set up the database. Refer to The PostgreSQL Global Development Group [121], PostgreSQL Global Development Group [96].

13.3.4 Populating the database with data

The database was populated with data by transferring the data files output from the MATLAB process model implementation, described in chapter 11, to the server and importing the data using the PSQL program.

The PSQL command *COPY FROM* is used to import data.

Refer to the PostgreSQL Global Development Group, PSQL Documentation in The PostgreSQL Global Development Group [121] which advises:

‘Use COPY FROM STDIN to load all the rows in one command, instead of using a series of INSERT commands. This reduces parsing, planning, etc. overhead a great deal. If you do this then it is not necessary to turn off auto commit, since it is only one command anyway.’

This command is only available to super users. An alternative for normal users is to use the *\copy* command available for PSQL. All the operations are stored in a *.psql* file which is executed by entering *\i LoadData.psql*.

Special entries for blank attributes and status values were added to cater for vertices or edges without attributes or status values.

The output generated by running *.psql* file is shown in appendix section J.9

A listing of the contents of the PostgreSQL database is given in appendix section J.6.

13.3.5 Server database verification

A trial version of the DBVisualizer database display program was used to view the database structure and contents as loaded on the desktop or server PostgreSQL application. Refer to Minq Software AB [84] for more information on DBVisualizer program.

Figure 13.5 shows the DBVisualizer summary display of the database tables.

The sample PostgreSQL database listing DBVisualizer print preview screen for the *tbperson* table of the Engineering Process Model database is shown in figure 13.4.

Appendix J.5 contains more details of the database structure.

The data on the server was made available in the Microsoft Access environment by defining linked tables in Access.

13.4 Microsoft Access Database Reference

As an alternative to referring to the database on the server the database was also set up using the Microsoft Access Database package. This provides the Engineering Process model database functionality in the desktop computer application environment on the Microsoft Windows XP operating system directly without any reference to data stored on a server. Refer to the software reference in Appendix J.7.

Access programming techniques were drawn from Litwin et al. [75, 76, 77].

13.4.1 Access data import process

The process model data is directly input into the Access database via Access Table links using a VBA program generating SQL queries for the DAO subsystem of Access. Refer to appendix section J.8 for more details.

The structure of the database reported by Access is shown in figure 13.6.

13.4.2 Database schema definition file

The external *schema.ini* file defines the record contents of the *.csv* data files imported into Access. An entry in the file for each table used in the database is required. Refer to appendix J.8.1 for a listing of the *schema.ini* file used.

13.4.3 Database program to import data from database server

Database Processing using Microsoft Access 1997/2002/2003

Data Definition Language (DDL) statements are a subset of the Structured Query Language (SQL) statements which include the SQL statements:

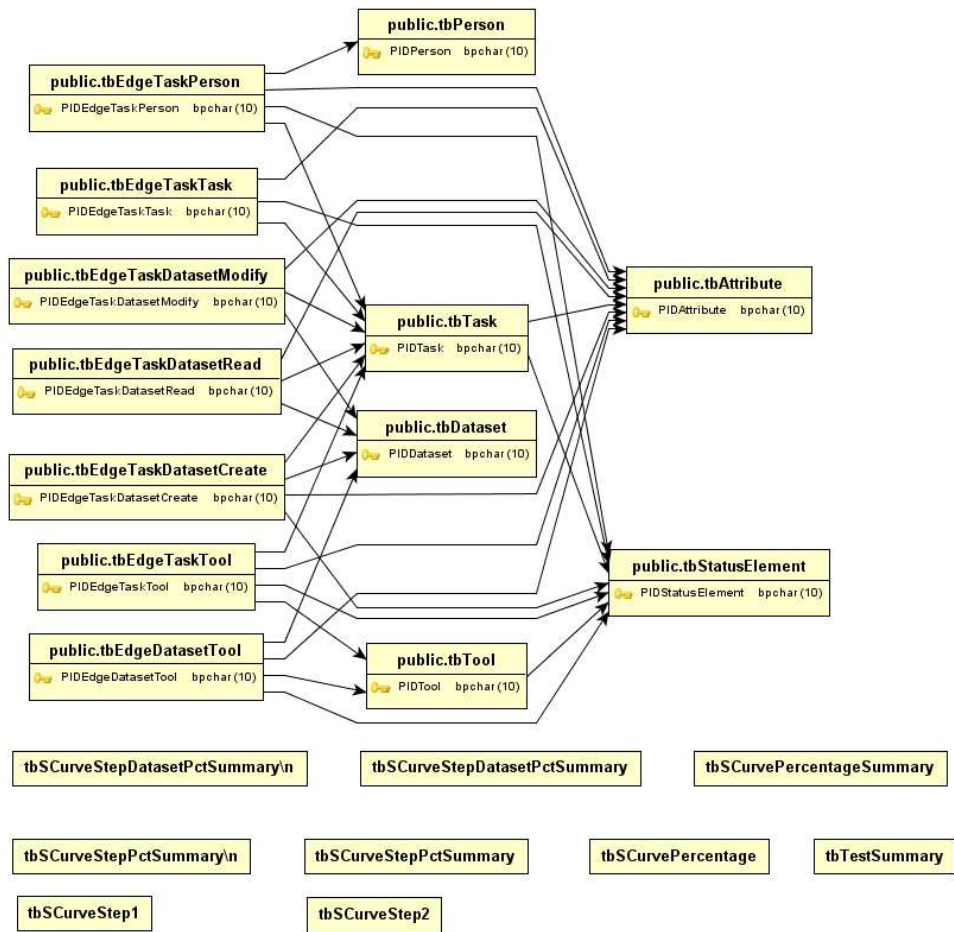


Figure 13.3: Process model database - DBVisualizer summary display

```
CREATE TABLE
ALTER TABLE
CREATE INDEX
CREATE INDEX
DROP TABLE
DROP INDEX
```

Access can accept DDL type queries in its query SQL view but can only execute one SQL statement at a time. This is a major drawback to importing DDL data from database design tools such as Azzurri Clay.

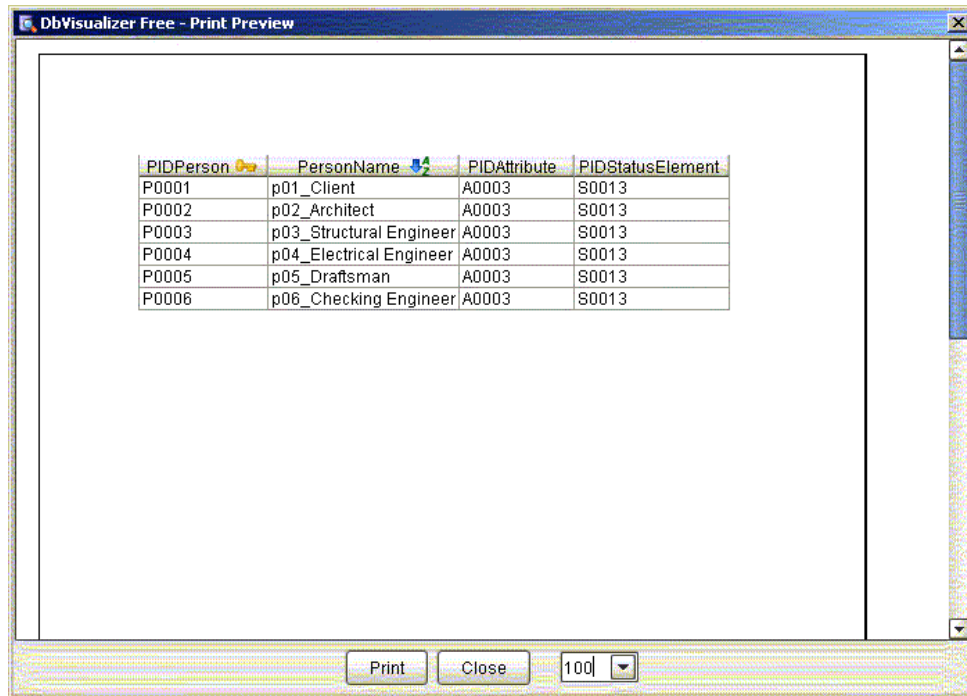
As an alternative, Visual Basic for Applications (VBA) programming using the Microsoft ActiveX Data Object (ADO) library and class/object model can be used as described in Litwin et al. [76, Chapter 5].

Refer to appendix section J.8.2 for a listing of the Access VBA program used to import data into the Access database directly.

Microsoft Windows Data Links

Microsoft Windows has a data link definition facility which can be used to set up a specification for a data link (data source) for use in Windows applications.

A blank text file is created using e.g. Windows Explorer, it is renamed to have a .udl extension and when the file is then opened it has the UDL functionality shown in figure 13.7.



The screenshot shows a window titled "DbVisualizer Free - Print Preview". Inside the window, a table is displayed with the following data:

PIDPerson	PersonName	PIDAttribute	PIDStatusElement
P0001	p01_Client	A0003	S0013
P0002	p02_Architect	A0003	S0013
P0003	p03_Structural Engineer	A0003	S0013
P0004	p04_Electrical Engineer	A0003	S0013
P0005	p05_Draftsman	A0003	S0013
P0006	p06_Checking Engineer	A0003	S0013

At the bottom of the window, there are buttons for "Print" and "Close", and a zoom level dropdown set to "100".

Figure 13.4: Process model database table print - DBVisualizer

The data content of the .udl file is shown in J.8.3.

The link can then be used in VBA programs to access data without specifying the link parameters in the program code.

Sample Microsoft Access VBA Code is given in appendix section J.8.3.

Refer to Litwin, Getz & Gunderloy - Access Developers Handbook Chapter 6, Litwin et al. [76].

Note that the Microsoft ADOX database access functionality will not work for all remote database servers. The Catalog/Table/Record structure need not be supported by all other databases e.g. PostgreSQL on Linux. Refer to the comment on PostgreSQL Web Site [96]:

‘Microsoft ActiveX Data Objects Extensions for Data Definition Language and Security (ADOX) is designed for use with the Microsoft Jet Database Engine. So, using ADOX with OLE DB providers other than the Microsoft Jet OLE DB Provider may cause unexpected behaviour or incorrect results. The exact behaviour is dependent on the nature of the database for which the provider is written. If a provider is accesses a database system whose model is totally different from that of Jet, the behaviour of ADOX could be unpredictable (for example, Jet does not support the concepts of CATALOG or SCHEMA) ...’

If errors are reported and the data link does not work, linked tables can be used.

It is important to note that Access DLL Library references set up in the software must be compatible as one moves from one Microsoft Windows installation to the next. Refer to figure 13.8 which shows the Access DLL reference display.

13.5 Reporting using Microsoft Access

Figure 13.9 shows a basic report generated using the reporting definition, display and printing functionality of Microsoft Access.

13.6 Reporting using Microsoft Excel

Figure 13.10 shows a typical Microsoft Excel tabular summary of the data processed by Access and imported into Excel, while figure 13.11 shows a chart defined in Excel referencing the Access data.

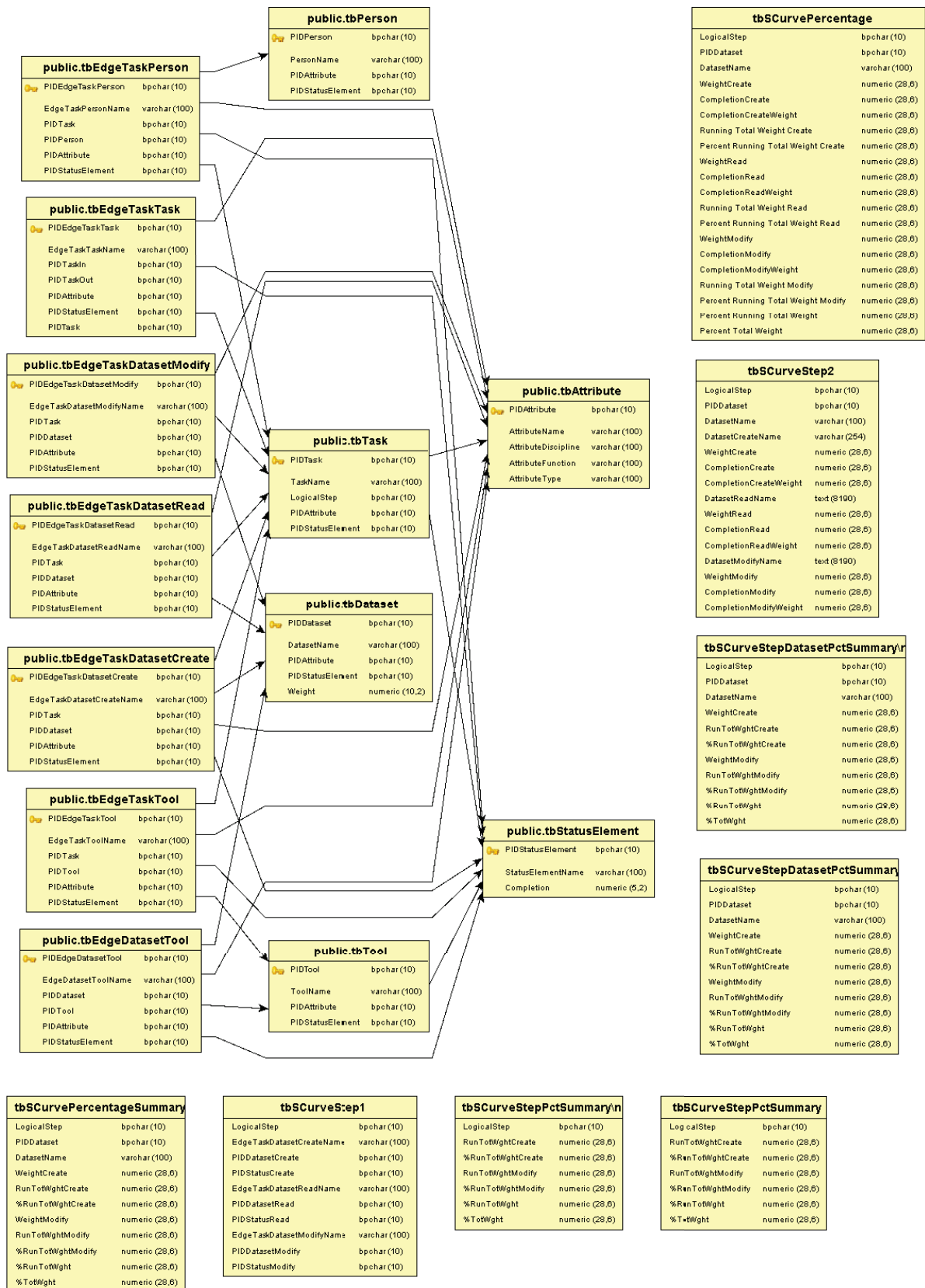


Figure 13.5: DBVisualizer extended display of process model database

Relationships for ODBCpSQLEngProcess
24 May 2006

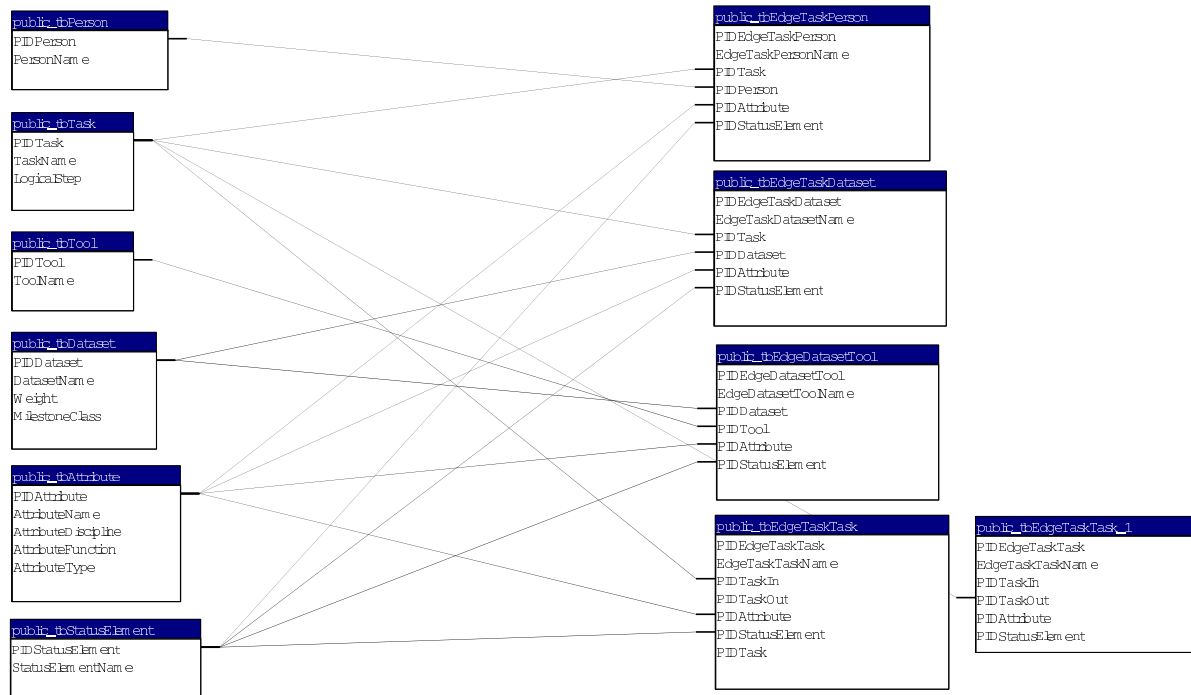


Figure 13.6: Access database structure report

13.7 PLEP application program for Engineering Process Model

The PLEP program is an implementation of the Engineering Process Model providing a user interface for data input as well as process optimisation. It was developed by Eygelaar [38].

The program was developed using the Java application programming language. Figure 13.12 provides an overview of the object data structure used in the program.

Functionality using e.g. the Java database connectivity (JDBC) to the Microsoft ODBC database access component (JDBC/ODBC bridge) can be used to export the data from PLEP to a database environment to supply database reporting functionality described in this document.

13.8 Importing database data using the Java JDBC-ODBC bridge

Java JDBC-ODBC bridge can be used for data loading. This is an alternative approach to populating database using files exported from the engineering process modelling application written in Java or the demonstration version done in MATLAB.

Refer to appendix section J.10 for the Java code of a sample implementation.

13.9 SQL Programming for reports and SQL functionality used

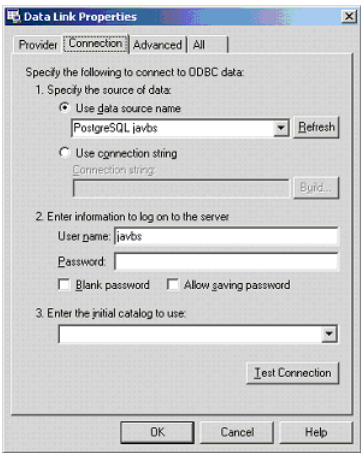
Although SQL standards like the ANSI-92 standard version has been defined, specific implementations of SQL all contain special versions with extensions or modifications of the standard. SQL contains conversion functions to convert data entries. Special care needs to be taken with null entries which can result from query operations due to null entries in database fields.

13.9.1 PostgreSQL conversion functions

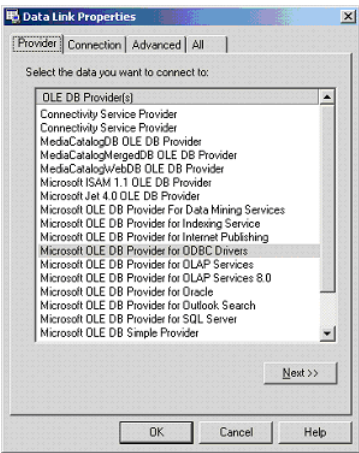
SQL conversion functions are described in e.g. Mata-Toledo and Cushman [80, page 151].

Conversions available include:

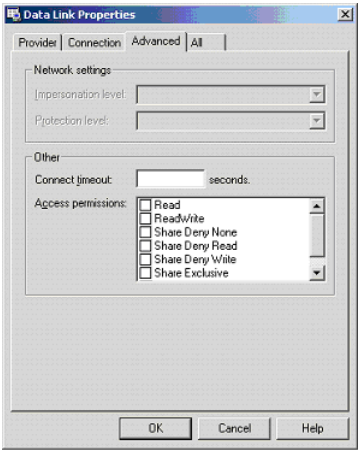
- null entries to string or numeric values - NVL(m,n) - Returns n if m is null else returns m



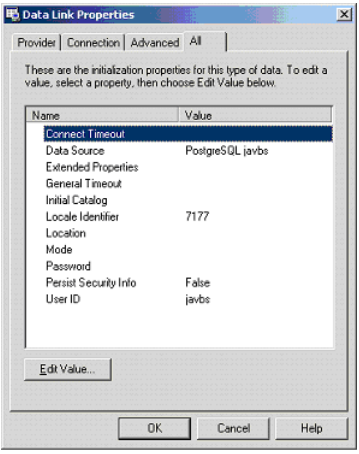
(a) Connection



(b) Provider



(c) Advanced



(d) All

Figure 13.7: Microsoft Windows Datalink tabbed panes

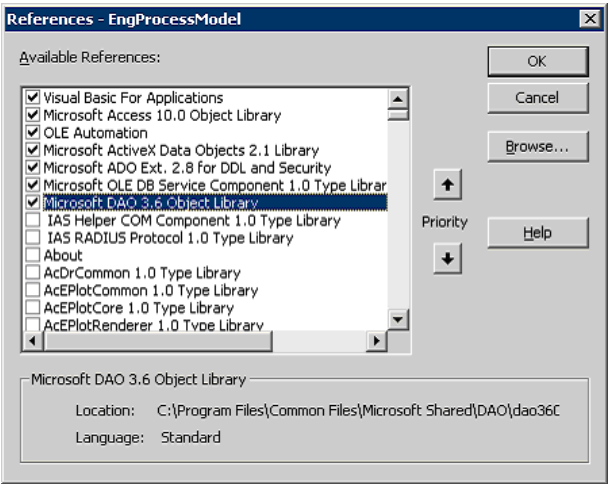


Figure 13.8: Access DLL library reference display

- numeric values into strings - `TO_CHAR(m[,fmt])` - numeric m converted from a number to a character string in designated format fmt

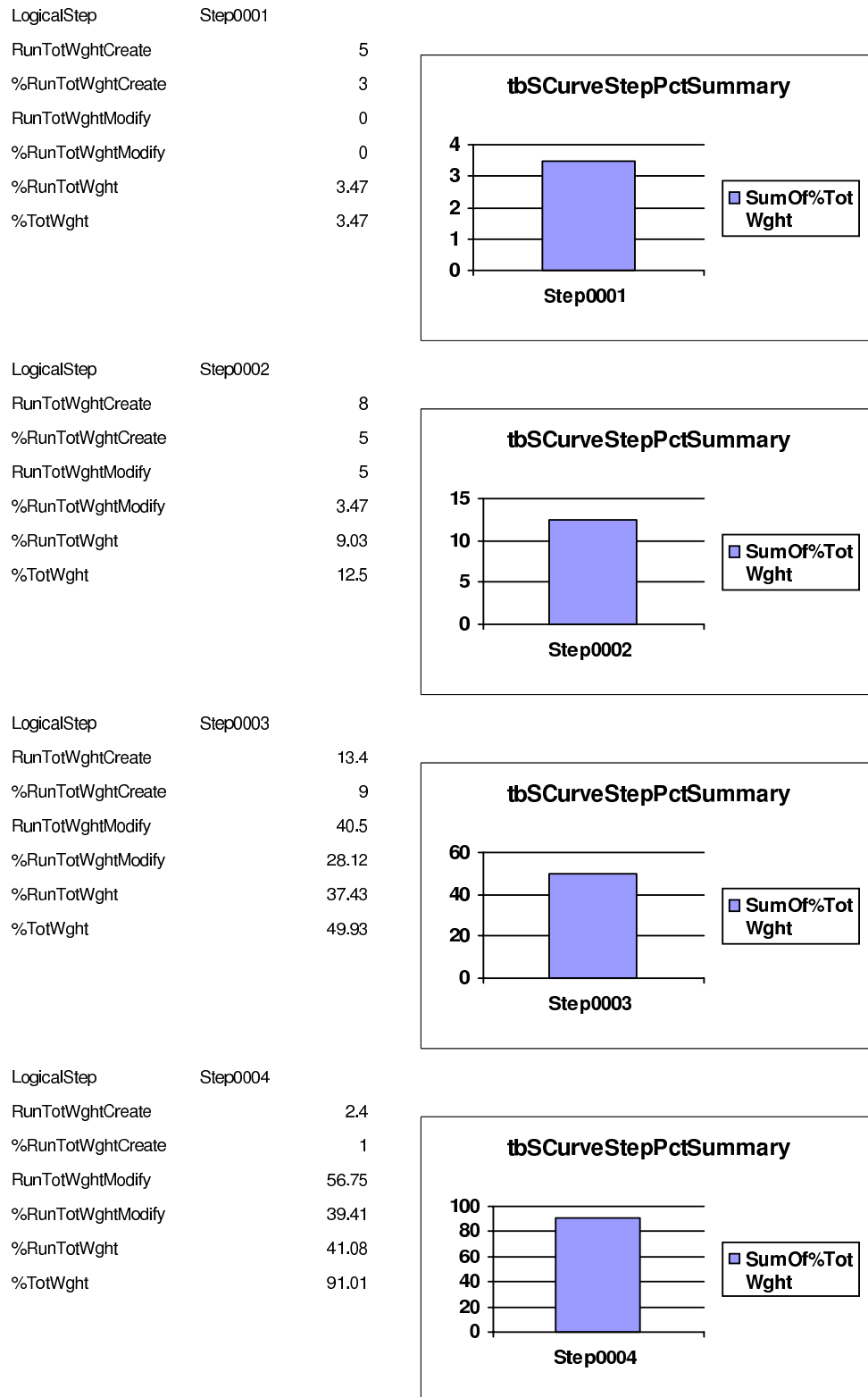


Figure 13.9: Engineering Process Example - Access Report

- string values to numeric equivalent - `TO_Number(st[,fmt])` - string st converted to a number according to designated format fmt
- rounding values to specific number of decimals - `ROUND(m,)`

The PSQL conversion options are very extensive - refer to the PSQL user documentation. The

tbSCurvePercentage

LogicalStep	PIDataset	DatasetName	WeightCreate	CompletionCreate	CompletionCreateWeight	Running Total Weight Create	Percent Running Total Weight Create	WeightRead	CompletionRead	CompletionReadWeight	Running Total Weight Read	Percent Running Total Weight Read	WeightModify	CompletionModify	CompletionModifyWeight	Running Total Weight Modify	Percent Running Total Weight Modify	Percent Running Total Weight	Percent Total Weight
Step0001	D0001	d01_Client requirements	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3.47
Step0001	D0002	d02_Architectural drawings	25	20	5	5	3	0	0	0	0	0	0	0	0	20	0	3.47	3.47
Step0001			0	0	0			0	0	0									3.47
Step0002								0	0	0			0	0	0				12.50
Step0002			0	0	0			0	0	0									12.50
Step0002	D0002	d02_Architectural drawings	0	0	0	5	0	0	0	0	0	0	25	20	5	20	3.47	3.47	12.50
Step0002	D0002	d02_Architectural drawings	0	0	0	5	0	25	0	0	0	0	0	0	0	20	3.47	3.47	12.50
Step0002	D0006	d06_Structural design report	25	20	5	24.4	3	0	0	0	0	0	0	0	0	97.6	3.47	6.94	12.50
Step0002	D0008	d08_Electrical design report	15	20	3	28.8	5	0	0	0	0	0	0	0	0	115.2	3.47	9.03	12.50
Step0003								0	0	0			0	0	0				49.93
Step0003	D0008	d08_Electrical design report	0	0	0	28.8	9	0	0	0	0	0	15	70	10.5	115.2	28.12	37.43	49.93
Step0003	D0007	d07_Electrical drawings	7	20	1.4	25.8	9	0	0	0	0	0	0	0	0	103.2	20.83	30.14	49.93
Step0003	D0006	d06_Structural design report	0	0	0	24.4	8	25	0	0	0	0	0	0	0	97.6	20.83	29.17	49.93
Step0003	D0006	d06_Structural design report	0	0	0	24.4	8	0	0	0	0	0	25	70	17.5	97.6	20.83	29.17	49.93
Step0003	D0004	d04_Concrete layout drawings	50	20	10	17	8	0	0	0	0	0	0	0	0	68	8.68	17.01	49.93
Step0003	D0003	d03_Foundation drawings	10	20	2	7	1	0	0	0	0	0	0	0	0	28	8.68	10.07	49.93
Step0003	D0002	d02_Architectural drawings	0	0	0	5	0	25	0	0	0	0	0	0	0	20	8.68	8.68	49.93
Step0003	D0002	d02_Architectural drawings	0	0	0	5	0	0	0	0	0	0	25	50	12.5	20	8.68	8.68	49.93
Step0003			0	0	0								0	0	0				49.93
Step0003	D0008	d08_Electrical design report	0	0	0	28.8	9	15	0	0	0	0	0	0	0	115.2	28.12	37.43	49.93
Step0003			0	0	0			0	0	0									49.93
Step0004	D0007	d07_Electrical drawings	0	0	0	25.8	1	0	0	0	0	0	7	75	5.25	103.2	38.37	40.03	91.01
Step0004								0	0	0			0	0	0				91.01
Step0004	D0008	d08_Electrical design report	0	0	0	28.8	1	0	0	0	0	0	15	10	1.5	115.2	39.41	41.08	91.01
Step0004			0	0	0								0	0	0				91.01
Step0004	D0006	d06_Structural design report	0	0	0	24.4	1	25	0	0	0	0	0	0	0	97.6	34.72	36.39	91.01
Step0004	D0006	d06_Structural design report	0	0	0	24.4	1	0	0	0	0	0	25	10	2.5	97.6	34.72	36.39	91.01
Step0004	D0005	d05_Reinforcement drawings	12	20	2.4	19.4	1	0	0	0	0	0	0	0	0	77.6	32.99	34.65	91.01
Step0004	D0004	d04_Concrete layout drawings	0	0	0	17	0	50	0	0	0	0	0	0	0	68	32.99	32.99	91.01
Step0004	D0004	d04_Concrete layout drawings	0	0	0	17	0	0	0	0	0	0	50	75	37.5	68	32.99	32.99	91.01
Step0004	D0003	d03_Foundation drawings	0	0	0	7	0	0	0	0	0	0	10	75	7.5	28	6.94	6.94	91.01
Step0004	D0002	d02_Architectural drawings	0	0	0	5	0	0	0	0	0	0	25	10	2.5	20	1.74	1.74	91.01
Step0004			0	0	0			0	0	0									91.01
Step0005			0	0	0								0	0	0				99.58
Step0005	D0003	d03_Foundation drawings	0	0	0	7	0	0	0	0	0	0	10	5	0.5	28	0.35	0.35	99.58
Step0005	D0004	d04_Concrete layout drawings	0	0	0	17	0	0	0	0	0	0	50	5	2.5	68	2.08	2.08	99.58
Step0005	D0004	d04_Concrete layout drawings	0	0	0	17	0	50	0	0	0	0	0	0	0	68	2.08	2.08	99.58
Step0005	D0005	d05_Reinforcement drawings	0	0	0	19.4	0	0	0	0	0	0	12	75	9	77.6	8.33	8.33	99.58
Step0005	D0007	d07_Electrical drawings	0	0	0	25.8	0	0	0	0	0	0	7	5	0.35	103.2	8.58	8.58	99.58
Step0005								0	0	0			0	0	0				99.58
Step0006	D0005	d05_Reinforcement drawings	0	0	0	19.4	0	0	0	0	0	0	12	5	0.6	77.6	0.42	0.42	100.00
Step0006								0	0	0			0	0	0				100.00
Step0006			0	0	0								0	0	0				100.00

Figure 13.10: Engineering Process Example - Excel tabular report

PostgreSQL Global Development Group [121, Chapter 9].

13.9.2 Microsoft Access data conversion

Access does not react to the rounding specification but has a *format* option in the properties of each column selected for a query as shown in figure 13.13.

Note that Action Queries (including SQL UNION queries) cannot be used as a row source i.e. saved as a table in Access. Refer to appendix section J.11.

The *Create Table* queries Access SQL uses *INTO tbName* and not *INTO TABLE tbName*.

13.9.3 SQL Query Processing tips

A copy of Access SQL queries are best kept outside the Access program in a text processing file format. The SQL data can then be copied and pasted to the SQL Query window as it is modified. Access reformats the SQL removing any indentation and blank lines as soon as the query is saved in Access.

Access SQL Query window does also not support the standard ‘-’ SQL comment line indication.

When setting up SQL statements e.g. SELECT entries, place the commas at the beginning of a subsequent line. This makes the *cut and paste* operation as queries are edited much less prone to syntax errors. Refer to the example below – (in PSQL format).

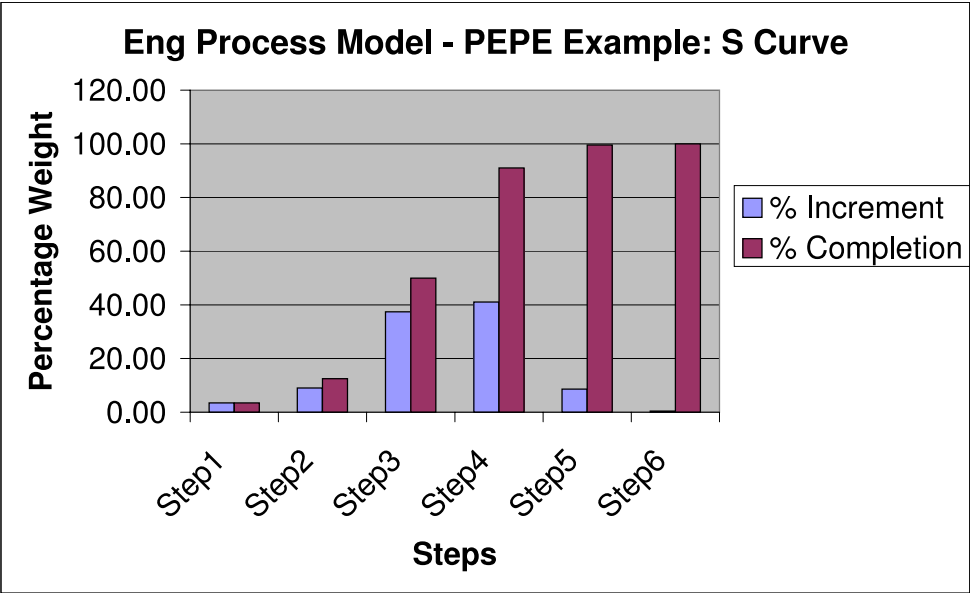


Figure 13.11: Engineering Process Example - Excel chart

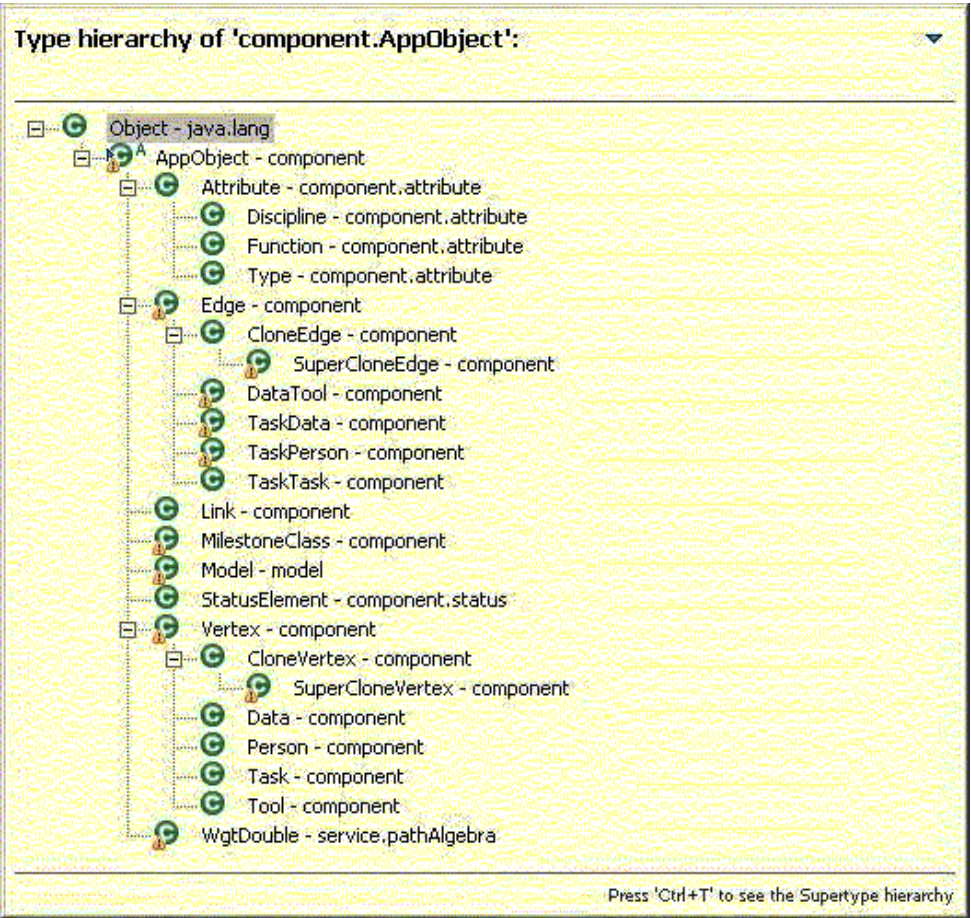
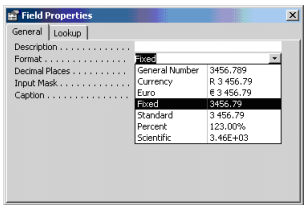
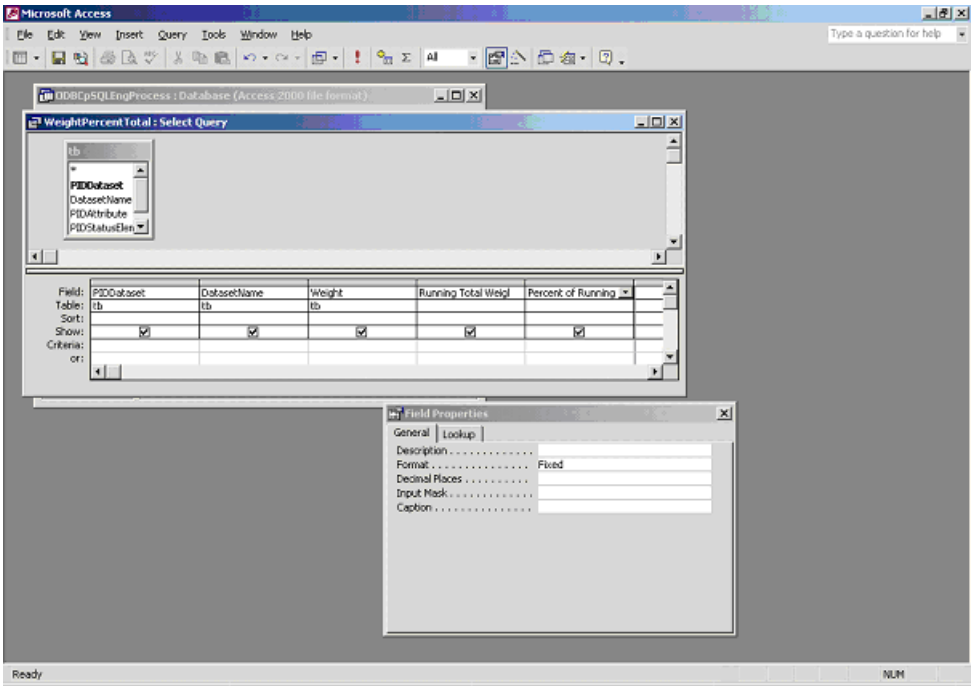


Figure 13.12: PLEP Java Application Object Structure gif



(a) General field properties



(b) Query definition pane

Figure 13.13: Microsoft Access Output Field Specification

```
SELECT  tb."LogicalStep"
        ,tb."PIDDataset"
        ,tb."DatasetName"
-- Data create values and percentages
        ,tb."WeightCreate"
        ,tb."CompletionCreate"
        ,tb."CompletionCreateWeight"
```

Note that in PSQL

DELETE FROM "tbSCurveStep2" WHERE "PIDDataset" = NULL; does not work

use: *DELETE FROM "tbSCurveStep2" WHERE "PIDDataset" ISNULL;*

13.9.4 SQL Queries for S-Curve presentation

The development was done in four environments:

1. Desktop standalone database application with data file imports
2. Linux application environment
3. Desktop client with Linux server environment with linked tables
4. Java JDBC-ODBC bridge environment to import data into the database environment and deliver the reporting or graphs to the Java application user interface

The following listings are given in J.11.1:

- the intermediate first step query for S-Curve recordset generation for the Access desktop client with PostgreSQL server
- the second step Access / PostgreSQL - SCurveStep2All.sql
- the file output in *.csv* format

The summary table output from Access is shown in figure 13.14.

tbSCurveStepPctSummary						
LogicalStep	RunTotWghtCreate	%RunTotWghtCreate	RunTotWghtModify	%RunTotWghtModify	%RunTotWght	%TotWght
Step0001	5	3	0	0	3.47	3.47
Step0002	8	5	5	3.47	9.03	12.5
Step0003	13.4	9	40.5	28.12	37.43	49.93
Step0004	2.4	1	56.75	39.41	41.08	91.01
Step0005	0	0	12.35	8.58	8.58	99.58
Step0006	0	0	0.6	0.42	0.42	100

Figure 13.14: Access database tabular reporting

13.10 Using Microsoft Data Access Pages

Data Access Pages can only be created using Access 2000 (or later) and can only be viewed by users of Microsoft Internet Explorer 5.0 (or later).

Any Access 2000 (and later) table, query and report can be saved as a data access page which is accessible as a form via the HTTP protocol.

The .htm file generated by Access for a data access page contains VBScript code.

There are a number of security issues with domains of users and file permissions which need to be set up carefully for remote users to pick up the data access pages which are in the form of scripts contained in .htm files.

The connection string in these files must contain the full name of the host reference for web browser (Microsoft Internet Explorer Version 5 or later) on a remote Windows PC, to display the data referred to in the access page.

Example: \\sivjavbs edited in place of C:\ provided by default data access page.

It seems as if the display in the browser does sometimes not work if the file is opened with the browser directly. When one double clicks on the file icon in the Windows Explorer display using the mouse, it opens in the browser and the display is shown.

A basic step progress report view in the browser environment is shown in figure 13.15 and figure 13.16

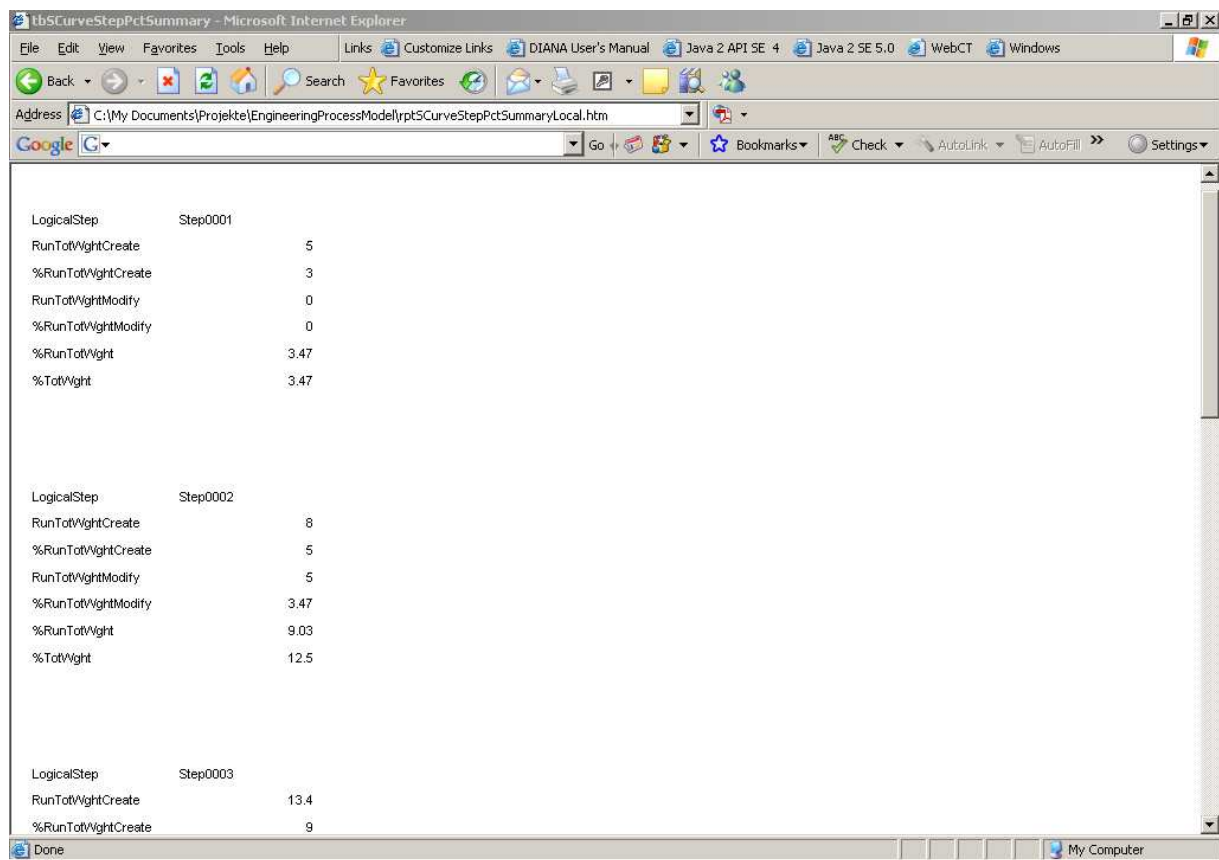
LogicalStep	RunTotWghtCreate	%RunTotWghtCreate	RunTotWghtModify	%RunTotWghtModify	%RunTotWght	%TotWght
Step0001	5	3	0	0	3.47	3.47
Step0002	8	5	5	3.47	9.03	12.5
Step0003	13.4	9	40.5	28.12	37.43	49.93
Step0004	2.4	1	56.75	39.41	41.08	91.01
Step0005	0	0	12.35	8.58	8.58	99.58
Step0006	0	0	0.6	0.42	0.42	100

Figure 13.15: Database tabular reporting with browser data access pages

The contents of the Data Access Page .htm file is listed in appendix J.12.

13.11 Conclusion and recommendations

Relational database technology in a client-server environment is a well developed and mature technology. The design of a database driven management reporting system needs to focus on the data structure and planned deployment to achieve the required level of system functionality. Interaction between desktop application software and database systems enhance the ease with which users can view and report data.



LogicalStep	Step0001
RunTotWghtCreate	5
%RunTotWghtCreate	3
RunTotWghtModify	0
%RunTotWghtModify	0
%RunTotWght	3.47
%TotWght	3.47

LogicalStep	Step0002
RunTotWghtCreate	8
%RunTotWghtCreate	5
RunTotWghtModify	5
%RunTotWghtModify	3.47
%RunTotWght	9.03
%TotWght	12.5

LogicalStep	Step0003
RunTotWghtCreate	13.4
%RunTotWghtCreate	9

Figure 13.16: Database reporting with browser data access pages

Part III

Conclusion

Chapter 14

Conclusions and Recommendations

The contents of this chapter aggregates and summarises the conclusions and recommendations of this dissertation. A summary of the conclusions of the system identification study contained in the addendum is included in chapter 27.

14.1 Conclusions

1. The theory set out in part I was applied in part II to develop technology which can be used in the development of engineering service enterprise management systems.
2. Techniques developed in part II can be applied to model business functions and management systems for the business functions identified in the addendum part IV of this study.
3. The review of the professional service business from a functional viewpoint reported in part IV of this study indicates that the basic technology demonstrated in part II is suitable as the basis of basic enterprise systems, management reporting and decision support systems for engineering services enterprises.
4. The Engineering process model (Chapter 11) with the graph tree based reporting structure (Chapter 12) can be adapted and applied to provide functionality for professional service business management.

14.2 Recommendations

1. Set theory, theory of relations and graph theory are not treated in current engineering curricula. Therefore a review of these theories was included in this document. The material covered in chapters 3, 4, 5, 6 and 7 will be of value to students and researchers working in the field of discrete mathematics applied to engineering.
2. The MATLAB functionality developed for this dissertation to implement the theory described in chapters 3, 4, 5 and 6 can be of value in the teaching of concepts in this field of study.
3. The present implementation of the Engineering process model which uses a step schedule needs to be extended to make time based, calendar linked, scheduling of tasks possible.
4. It should also be possible to apply the path algebra theory set out in chapter 5 to develop the reporting structures set up using graph adjacency matrix manipulation.
5. The development of a user interface for the graph theory based management reporting structures which can be used to generate SQL code and link to a database can be attempted in future.
6. A full implementation of an enterprise management (ERP) system fell outside the scope of this dissertation, but can be based on the concepts developed here. The implementation of an ERP system for a typical professional practice, based on the information contained in this study, can be considered.

Part IV

Addendum: Identification of system functionality to provide support for management functions

Chapter 15

Overview of Part IV

This part forms an addendum to the thesis document.

The basic business functions applicable to the professional service business are used to identify aspects which impact on business systems and business management system functionality.

Reference is made to the techniques and technology developed in the previous parts.

Business models which are defined in this part contain business objects and business processes. Selected attributes of models and processes can be identified for database processing and management reporting hierarchies. Techniques and technology developed in the previous parts can be used to process information on the business object and process logic identified.

An indirect outcome of the analysis is the provision of a high level specification or *road map* for the development of a flexible Enterprise Resource Planning (ERP) system for the professional service business.

According to Wikipedia [126], ERP systems integrate (or attempt to integrate) all data and processes of an organisation into a unified system. A typical ERP system will use multiple components of computer software and hardware to achieve the integration. A key ingredient of most ERP systems is the use of a unified database to store data for the various system modules. ERP systems typically attempt to cover all basic functions of an organisation, regardless of the business or charter of the organisation.

The professional service business functions reviewed include:

- Business strategy and long term planning and general management
- Marketing, promotion and public relations management
- Finance including bookkeeping and auditing
- Personnel and personnel management
- Production management, i.e. the management of the execution of project work
- Facilities management and document management
- Knowledge and information management
- Logistics, i.e. the management of resource required for the business to operate including knowledge management
- Administration
- Risk management

An overview of the typical professional practice with its associated management systems is shown in figure 15.1.

A brief overview of commercially available software packages which can be implemented for consulting engineering enterprise management is also given.

Conclusion, recommendations and suggestions for further work

The review of the professional service business from a functional viewpoint reported in this part of the study indicates that techniques developed in the previous parts can be applied to model business functions and management for the functions identified here.

The engineering process model (Chapter 11) with the graph tree based reporting structure (Chapter 12) can be adapted and applied to provide functionality for professional service business management.

The implementation of an ERP system for a typical professional practice, based on the information contained in this study, can be considered. However, this falls outside the scope of this study.

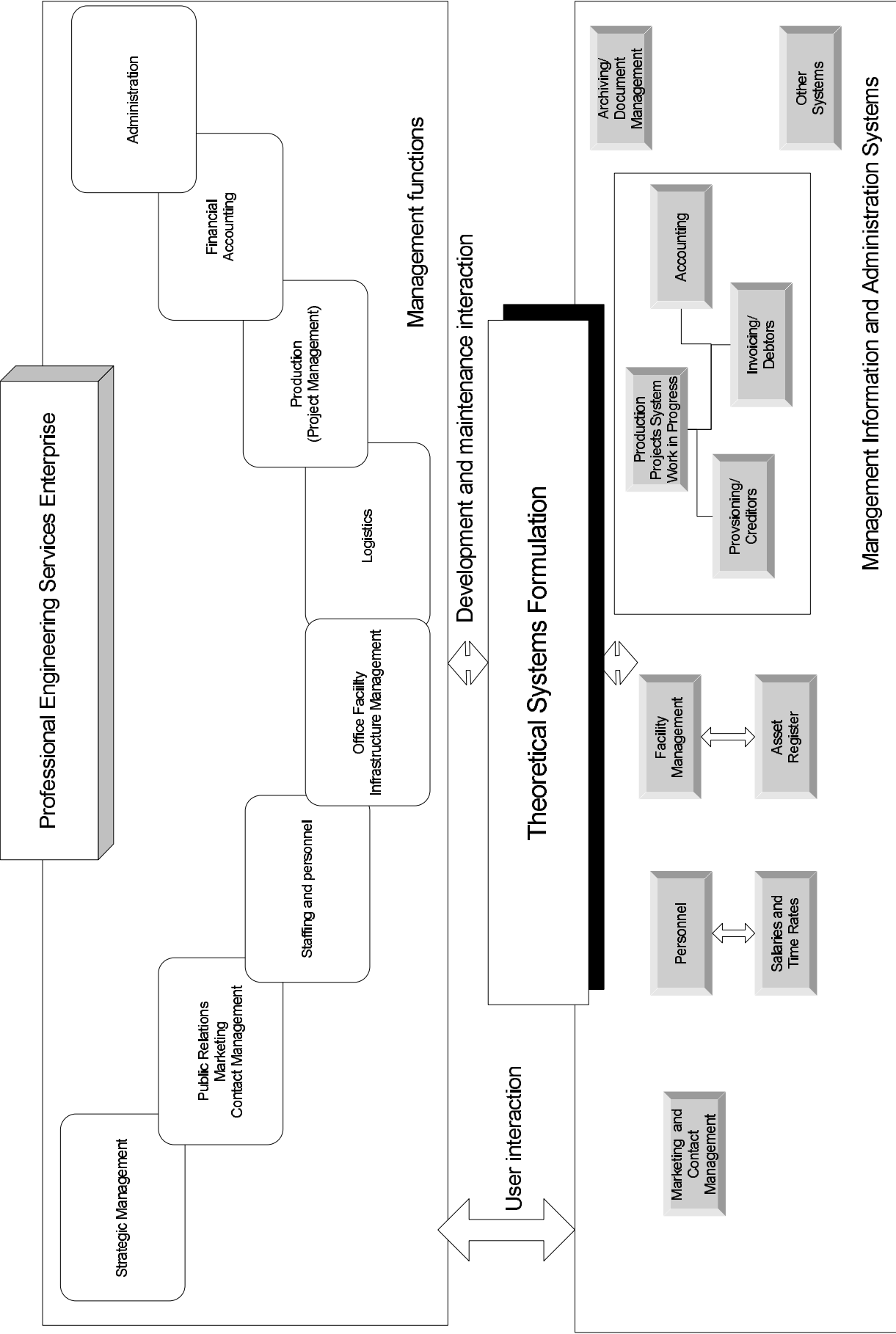


Figure 15.1: Systems Theory Based Professional Engineering Services Enterprise Management Information Systems

Chapter 16

Business strategy, long term planning and general management

A review of business strategy, long term planning and general management concepts are described in this chapter.

Reference to books and documents by Michel Robert on business strategy [102], Decision Process International manuals [31], Wim de Villiers on management in general [28] and Kenneth Barlow on Professional Management for Consulting Engineers and Architects [12] was made in the process of compiling this chapter.

16.1 Business Strategy Concepts and Strategy development

This section reviews thinking on business strategy and related concepts developed by Michel Robert. It provides a concise introduction to this field of knowledge and expertise.

16.1.1 Elements of a business which reflect strategy

Michel Robert identifies the major elements which impact on business strategy and long term planning as:

- Nature of products
- Nature of customers and groups of customers
- Nature of market segments
- Nature of geographic markets

These four elements are a key part of the future strategic profile or vision of a business. Strategic statements dealing with the elements need to be stated in both positive and negative terms i.e. *focus on* or *not focus on*. Defensive and offensive strategic objectives, with associated strategies, can be devised.

16.1.2 Physical indicators of the direction and ‘look’ of an enterprise

The present direction and organisation is following can be deduced by studying:

- The product catalogue both present and future
- People and skills which management are trying to draw into the organisation
- Markets served
- Competitors
- Customers
- Suppliers
- Market segments
- Research and Development budget
- Facilities

16.1.3 Strategic areas comprising an organisation

The strategic areas comprising an organisation are identified as:

- Product/service concept
- User/customer class
- Market type/category
- Production capacity/capability
- Technology/know-how
- Sales/marketing method
- Distribution method
- Natural resources
- Size/growth
- Return/profit

A strategic selection of an area which is the driving force of the enterprise needs to be made. Areas of excellence need to be cultivated and describable skill competence or capability in these areas need to be cultivated.

16.1.4 Maintaining a strong and healthy strategy

Table 16.1 outlines the focus of excellence development for the listed strategy concepts.

Table 16.1: Maintaining a strong and healthy strategy - Robert [102]

Strategy concept	Focus of excellence development
Product/service	Product or service quality Product and process development Excellence in product development and service
User/customer class	Market and user research User loyalty
Market type/category	Market and user research User loyalty
Production capacity/capability	Optimising manufacturing or plant efficiency Marketing to Substitute other products
Technology/know-how	Research Marketing to find applications of new products
Sales/marketing method	Recruitment of sales personnel Improving effectiveness of selling methods
Distribution method	Distribution method and effectiveness Optimise and improve distribution effectiveness
Natural resources	Exploring to find new resources and sources of resources
Size/growth	Financial management and portfolio management Information systems
Return/profit	Financial management and portfolio management Information systems

A direct summary of strategic focus is outlined in the Johnson & Johnson Credo which reads:

First responsibility is to our customers
 Second responsibility is to our employees
 Third responsibility is to our community
 The last responsibility is to our shareholders

16.1.5 Articulating the business concept of the enterprise

To articulate and communicate the business concept of the enterprise, the vision or strategic profile (synonyms) as well as the strategy, business concept, mission, mandate, charter (all synonymous) need to be formulated.

The business driving force and strategic heartbeat needs to be articulated in concise terms.

Strategic statements dealing with the:

- scope of products
- scope of customers and markets
- organisation structure
- technologies required
- type of production facilities
- distribution channels
- marketing and selling techniques
- type of personnel employed

need to be formulated.

People do not implement what they do not properly understand and do not implement what they are not committed to.

Corporate competition dictates that successful companies need to leverage their unique set of capabilities i.e. driving force and areas of excellence across the largest number of products and markets.

16.1.6 Operational objectives

As part of the strategy formulation exercise, formulation of goals, projections and budgets for the major business functions are required. These functions include:

- Sales
- Marketing
- Manufacturing
- Accounting
- Human resources
- Research
- Customer service

16.1.7 Developing strategic business models

An analysis of the concepts outlined above indicates that strategic business objects can be identified and a hierarchy linked to business activities and processes can be defined. This structure can form the basis of database queries where selected attributes can be included in management reports, as shown in chapters 13 and 12.

The logic embodied in table 16.1 can be used as input for goal based reporting structures which align business functions with strategic and lower level goals. Refer to Scheer [108, pp 3 and 23].

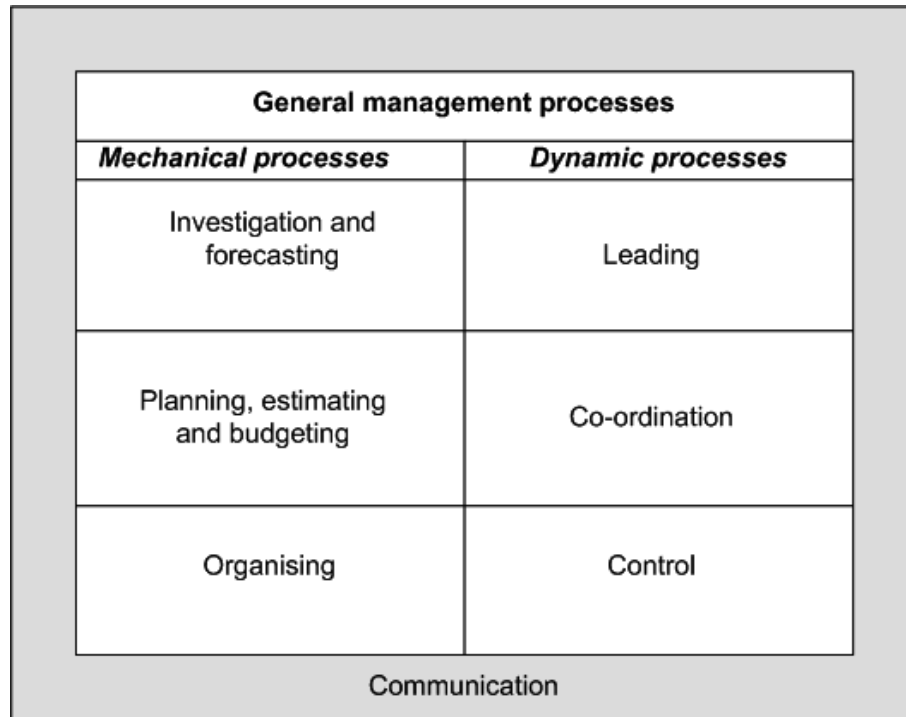


Figure 16.1: General Management Aspects according to De Villiers [28]

16.2 General Management

This section contains a review of an approach to general management concepts developed by Wim de Villiers for the Gencor organisation. [28]

Aspects (processes/activities) of general management grouped into ‘mechanical’ aspects and ‘dynamic’ aspects.

Figure 16.1 contains an adapted version of the general management concept developed by De Villiers.

Investigation and estimating, planning and organising are classified as ‘mechanical’ aspects seeing that they provide *form* to the management structure and processes.

Leading, coordinating and controlling are classified as ‘dynamic’ aspects of management because of the *functional nature* of the processes and activities. These processes are applied on a day to day, ongoing basis in management.

Communication is a process which links management to parties including the organisation itself as well as its environment for the transfer of information as required.

16.2.1 Mechanical aspects of general management

The mechanical aspects of general management are set out in this section.

Investigation and forecasting

To fulfil its investigating and forecasting role management needs to:

- Formulate issues and problems and set goals.
- Collect and collate information internal and external to the organisation and do Strengths, Weaknesses, Opportunities and Threats (SWOT) type investigations.
- Analyse the information in a two step process focusing on the past and present and then on the future doing the necessary forecasts and estimates.
- Decide on a solution – What is to be done? Develop and evaluate alternatives and select a suitable course of action.

Planning, estimating and budgeting

The business policy (vision, mission) provides the framework for the enterprise. Management planning, estimating and budgeting activities need to interpret the framework and set up the:

- High level strategy and policies - in detail
- Strategic plan and planning framework
- Five year down to one year, financial plans and quarterly and short term planning estimates and budgets

Management then needs to:

- Decide on actions and the approach required leading from the investigation i.e., how is it going to be done?
- Do a detail investigation and develop detail plans which define actions to be taken.
- Formulate and communicate plans using programming, scheduling and budgeting techniques.
- Develop controls for global and detail plans consisting of standards to be applied, measurement and corrective action.
- Develop contingency plans based on risk analysis and risk management principles.

Organising

The organisation reference framework is made up of the:

- Organisation structure
- Manpower planning, supply and management succession plan
- Remuneration and personnel care
- Education and training
- Industrial and personnel relations
- Processes in the organisational development

Typical steps required for organising are:

- Analyse the tasks at hand and determine manpower and other resources required to achieve goals. Logically group and structure tasks and activities.
- Delegate authority to personnel in appropriate posts by determining the type of work to be done, the scope of the authority to be delegated and coordinating and controlling activities required for each post or job.
- Set up the relations between posts in a logical hierarchy and define interaction between personnel for each post or job.
- Determine the human characteristics required per post and staff posts by personnel search and selection. Educate and train personnel for progress to higher level posts or jobs.

16.2.2 Dynamic aspects of management - activities and processes

The procedural framework defines processes, activities and procedures which need to be dealt with under the dynamic aspects of management i.e. leading, coordinating and controlling. These are:

- Production
- Procurement
- Stock inventory and material control

- Personnel
- Training
- Safety
- Security and risks
- Other

The dynamic aspects of general management are set out below.

Leading activities and processes

Leading, in the management sense of the word, covers activities which management are involved in to communicate with, educate and train personnel and to delegate tasks to suitably motivated persons in the organisation.

Leading also includes feedback on work done, the handling of problems and grievances, dealing with unacceptable behaviour and substandard work and the taking of corrective measures.

Coordinating activities and processes

The coordinating aspect of management deals with the day to day interaction between personnel and management to ensure that the work at hand gets done in an efficient manner. Efficient communication and office, department and project meetings as required should solve most coordination issues. The process of dealing with ad-hoc events which influence the business can also be seen as a coordinating activity.

Coordinating activities and processes focus on both mechanical as well as dynamic aspects of management.

Coordinating to ensure application, effectiveness and efficiency of 'mechanical' aspects of management requires:

- Clear goals based on policy guidelines
- Clear and concise plans, programmes, organisation diagrams and budgets
- Ensure that personnel grasp the organisational structure and procedures

Coordinating to ensure application, effectiveness and efficiency of 'dynamic' aspects of management requires:

- Participation in decision making
- Regular adjustment in plans, programmes and budgets, as required
- Effective communication in the organisation

Controlling activities and processes

The framework for control contains processes for control and information dissemination on production control and costing control.

Dynamic management aspects of control include setting goals with personnel and reviewing performance with personnel.

Steps for setting goals with personnel are:

- Explain the why and how of the task
- Agree on measurable goals: what, when and how much
- Ask for input for proposals to achieve goals and expand ideas
- If required, agree on additional goals and improved standards
- Offer help and confirm trust in personnel member
- Agree on follow-up as far as actions and dates are concerned

Steps for controlling performance with personnel are:

- Confirm goals agreed to
- Call for a progress report and note success achieved
- Ask for steps for improvement and supplement ideas
- If required, agree on additional goals and improved standards
- Offer help and confirm trust in personnel member
- Agree on follow-up as far as actions and dates are concerned

16.2.3 Communication

Communication is shown as a wrapper of the other processes shown in figure 16.1. This indicates the importance of management communicating the form (objects) and function (processes or activities) of management objectives and actions to all parties and stakeholders, including the members of the organisation itself, as well as its environment.

Management needs to communicate aspects of the ‘mechanical’ as well as ‘dynamical’ aspects of management to role players inside and outside the organisation as required. This will ensure smooth operation of the business enterprise.

The basic principles of good communication and well honed listening skills need to be applied here.

16.3 Conclusion and recommendation

The author was involved in a long term project to implement the strategic, functional and management aspects of a Water Plan for a local authority. Application of techniques developed with reference to the material covered in this chapter proved to be well received by management and staff at all levels of the local authority involved in the project. Refer to Strasheim et al. [118] for more information on the approach adopted for this project.

Strategic business planning and reporting structures can be developed and metrics needed to supply high level feedback on the ‘health’ of a business as whole can be planned in a system format suitable for implementation, using the theory covered in part I and II of this thesis.

Chapter 17

Marketing, promotion and public relations management

17.1 Introduction to professional services marketing management

This chapter deals with the management of the marketing function of the engineering professional services enterprise.

The aim is to provide the background information and system analysis, synthesis and system building insights to lead into the business objects and processes needed for modelling enterprise marketing activities and the management thereof.

The business objects and processes to define models for marketing management in this context are identified.

The differences between product and services marketing are highlighted and the nature of professional services marketing defined.

Aspects of promotion and public relations which link to marketing management are discussed.

The book by Young [129] covers marketing of the professional services firm and includes a section on tools for marketing services.

17.2 Differences between consumer product marketing and professional services marketing

This section deals with the nature of marketing as well as the mechanical and dynamic aspects of marketing management. Mechanical aspects of marketing are:

- marketing investigation
- environmental scanning
- forecasting
- marketing planning
- marketing budgeting
- organising for marketing.

The dynamic aspects of management of marketing are:

- marketing leading
- coordinating of marketing activities
- and controlling of marketing activities.

It is important to highlight the major differences between consumer product marketing and professional services marketing. Professional service marketing needs to be viewed and approached in most of its aspects, discipline, body of knowledge and best practice much more like industrial marketing than consumer goods marketing. Wittreich [127] summarises major differences between the marketing of services and products. The differences are shown in table 17.1.

Table 17.1: Differences between services and product marketing

<i>Aspect</i>	<i>Product</i>	<i>Service</i>
Evaluation for purchase	Typically a physical sample is available for evaluation	Persons and groups of suppliers supplies proposals describing service to be sold
Customer notion of risk and uncertainty and confidence in what is being bought	Low risk and high confidence by referring to specifications and available usage data	Buyer in hands of seller
Alternatives	Limited range of well defined alternative products	Service on offer can be modified / expanded based on information supplied by customer

17.3 The nature of professional services marketing

Marketing is a contact based activity. According to Warner [125] personal contact must be regarded as the single most important professional services marketing function.

According to Wittreich [127], three concepts are fundamental to marketing of professional services.

- The goal of the professional service organisation is to identify aspects of uncertainty in the business of the client, where the service is to be provided and bring about an increased degree of certainty where uncertainty is felt.
- When management considers the purchase of a professional service it should insist that the representatives of the service providers be able to address the substantive problem of the client directly.
- Management should insist on dealing directly with individuals of true professional competence who must be capable of rendering the service.

Typical types of uncertainty on the client (procurer) side which need to be addressed by the service bidder or provider are:

- Who should supply the service? The supplier needs to show that an unique, quality service is being offered.
- Is value for money being offered? The sale is basically being closed as the promised service is being delivered by the supplier.
- Are the substantive requirements of the client being defined in meaningful terms? The professional service provider needs to be able to listen and analyse and integrate the problem and requirements of the client into a logical whole. Real issues need to be identified and if the client does not have a real problem, the seller needs to indicate this. Professional ethics and integrity on the side of the seller needs to apply at all times.

Buying a service is analogous to hiring a key employee in scope and impact on the business of the client. However the attributes of key personnel of the service provider are not necessary and complete requirements to a successful outcome for the rendering of the service. Selling by focusing on success stories through analogy should have more impact on the decision making of the client.

When professionals are identified to render a service, demonstrable knowledge and skill in areas of applicable competence need to be shown and recognition of limits of knowledge and skills on the side of the supplier needs to be conveyed where applicable.

A statement by Robert [102] on strategy comes to mind here: ‘Developing a describable skill competence or capability in a company to a level of proficiency better than anything else it does and particularly better than any competitor does’.

Young [129, page 25, Table 1.2] identifies a number of unique qualities of services which need to be taken account of in the marketing of these services.

- *Intangibility*: Services are intangible and consist of an action or deed
- *Inseparability*: The buyer of a service finds it difficult to distinguish between the service provider and the service.

- *Simultaneous consumption and perishability*: Service are consumed as they are produced and cannot be stored, saved, returned or easily changed
- *Variability*: Services are very difficult to standardise
- *Ownership*: The work and outputs of the work of the service supplier are bought but not the supplier and his resources
- *Process*: Services dictate a process through which clients must pass

Young [129] reproduces a figure from Leonard Berry showing the nature and roles of service marketing, shown here in figure 17.1.

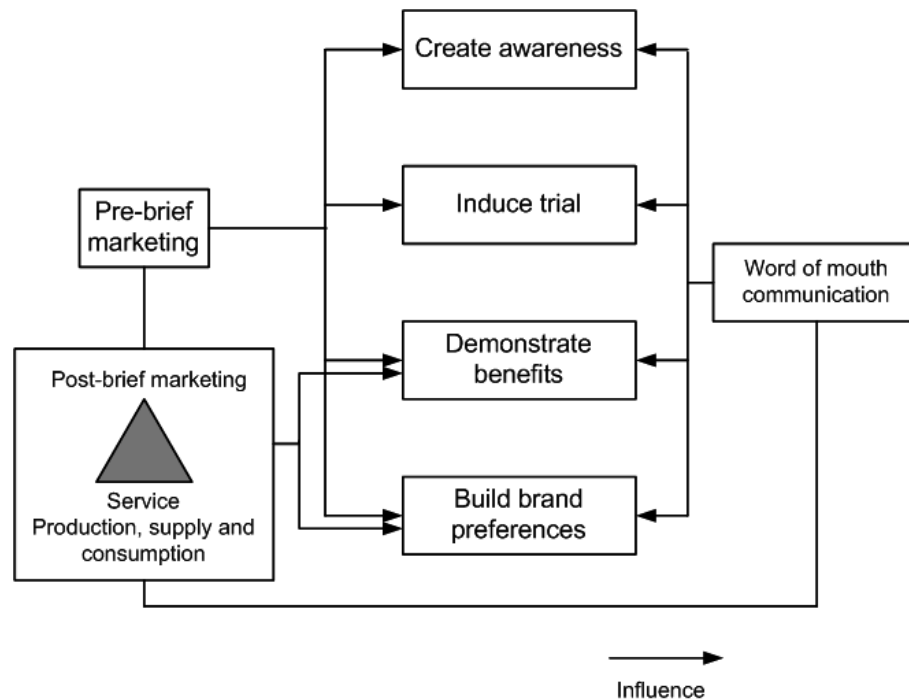


Figure 17.1: *The nature and roles of service marketing according to Leonard Berry*

17.3.1 Business models for marketing management

Business models for professional service marketing management can be constructed by referring to business objects for marketing management and business processes relating to marketing management.

17.3.2 Business objects relating to marketing management

The listings below contain a selection of typical objects and object classifications for professional services marketing management.

Basic marketing model logical objects:

- Marketing person, group or office and department
- Client person, group or business
- Project proposal document
- Pro-forma client brief and contract
- Calendar recording marketing periods, events and history
- Marketing events such as:

- Personal contact sessions or meetings
 - Conferences
 - Sporting and entertainment events
- Marketing material such as:
 - Promotional letters and brochures
 - Personnel *curriculum vitae* documentation
 - Pricing and fee schedules
 - Company newsletters
 - Press releases
 - Media programmes in voice or video format

Marketing management documents:

- Environmental scan and competitor analysis reports
- Strategic marketing plan
- Marketing planning sheet
- Public sector client consultant panel and preferred service supplier databases
- Marketing status report sheet
- Client database
- Project database
- Marketing budget with projected project cash flow

(Refer to appendix L for typical sample documents.)

17.3.3 Business processes relating to marketing management

The listing below contains a selection of processes for professional business marketing management.

- Market research and market analysis
- Competitor analysis
- Marketing planning
- Budgeting for marketing
- Customer relationship building
- Advertising and promotion
- Business alliance formation
- Client follow-up and project wrap-up meetings
- Control of marketing activities

Some of these aspects are now discussed in more detail.

17.4 Marketing investigation, environmental scanning and forecasting

This section describes activities and processes leading into and supporting the formation of marketing strategies and marketing planning.

17.4.1 Corporate/enterprise requirements

An important part of marketing is to ensure that what is to be done ties in with the strategy, vision and mission of the enterprise. Higher level objectives of the organisation and its units also need to be considered. Basic budgetary process requirements and constraints need to be taken into account when the aspects of marketing dealt with below are considered.

17.4.2 Environmental scanning and forecasting

Environmental scanning and forecasting is a wide ranging and specialised activity. Professional support can be considered if an enterprise needs to undertake this kind of activity. Well organised and controlled in house reference to published literature and media content should meet most of the requirements in this field for a typical services enterprise.

Enterprise/employer organisations such as the South African Association of Consulting Engineers (SAACE) and South African Federation of Civil Engineering Contractors (SAFCEC) typically provide the outcome of cooperative systematic and ongoing marketing research to member enterprises belonging to these organisations.

It might be necessary to embark on activities in this ambit where specialised requirements need to be met and employ specialist consultants as required.

Business intelligence encompasses the concept of obtaining and analysing information on existing clients as well as possible future clients. An example of the type of analysis which can be undertaken is that of the development of a process model of the processes of a government department division. Strasheim [116] reports on a review and model of the business process (development project work flow) of the Community Water and Sanitation Services (CWSS) programme division of the Department of Water Affairs and Forestry (DWAF).

17.4.3 Marketing research and market research

Elements of marketing research identified by Du Plessis [32] are:

- research into services to be offered
- research into markets. Size, share, regional breakdown, market forecasts, client policies, attitudes and preferences
- marketing intelligence and research into competitor, supplier, client activity (industrial espionage)
- research into marketing methods and practice
- methodology for marketing research: problem statement, research design, sample selection, data collection and analysis and research reporting with presentation.

Note that market research is seen as being an element of marketing research.

The note in paragraph 17.4.2 on the SAACE and SAFCEC activity in this field also applies here.

17.4.4 Market segmentation

To focus marketing activities it is important to do some form of market segmentation before embarking on marketing activity planning and budgeting.

As an example, the market can be divided into government institutions, private developers, private persons as well as indirect exposure via other professionals e.g. architects, civil engineering consultants, mechanical engineering consultants and quantity surveyors.

17.5 Strategic planning for marketing

A market sector analysis should indicate market sectors with activities and sectors where capital expenditure on projects is more likely in the medium term.

An existing and potential client hierarchy can be developed.

Linking up with other professionals e.g. architects, town and regional planners, electrical and mechanical engineers, quantity surveyors and landscape architects to exchange marketing information and form groupings in approaching clients can be valuable.

The present government procurement requirements in South Africa dictate shareholding requirements for professional organisations which do government work on all three tiers of government. It might be required that an enterprise restructure its ownership or form new ventures to meet the requirements set by government.

Cross marketing, i.e. referring projects to other entities of the enterprise or related group companies for follow up can be considered. As an example, infrastructure and facilities management projects and activities can be used to identify potential new upgrade or refurbishment projects.

17.6 Marketing activity planning and budgeting

An example of a hierarchy used for marketing activity planning is shown in appendix L, figure L.1. Note that the activities listed here exclude that of project marketing which goes with the development of project proposals and the activities which lead up to a brief being issued by a client.

Puttergill [99] divides business development (marketing) into the areas of corporate development, general marketing and project related marketing.

Detailed marketing planning leads to data gathering and analysis for reports such as the one shown in appendix L, figure L.3.

17.7 Project phases

This author is of the opinion that the administrative procedure of registering a project as soon as a contact for a possible project has been made and defining the status of the project as ‘Proposal’ to be changed later as required and closing the project if the proposal is not successful, is the preferred approach to marketing planning and administration. This approach is also advocated by Robertson [103] and is implemented in the ProMan professional practice management software package Greyling [50].

Puttergill [99] classifies projects as being *in hand*, *in view* or *to be obtained*.

After project closure contact with existing clients should be maintained.

17.8 Organising for marketing

With reference to the GFJ Inc. document on a marketing strategy by Warner [125] as well as appendix L, typical activities which are organised for professional services marketing are:

- Visit an existing client or a potential client
- Development and maintenance of contact and client databases
- Organising of special corporate events
- Organising of symposia which can be sponsored events
- Presentation of client and corporate entertainment events
- Preparation of technical brochures
- Development and maintenance of technical personnel curriculum vitae data sets
- Organising and participation in institutional activities e.g. SA Institution of Civil Engineer, SA Association of Consulting Engineers, SA Institution of Municipal Engineers
- Write an article for a trade magazine or rework an existing article for publication in another magazine
- Write a conference contribution or article for publication in a professional technical magazine with peer review
- Develop and publish a company newsletter on a regular basis
- Develop and maintain information and data required to be on record
- Prepare and publish press releases and other media exposure.

17.9 Marketing leading

Enterprise owners, shareholders, executives and managers need to be seen taking a leading role in all marketing activities and ensure that junior personnel are informed of activities as required.

17.10 Coordinating of marketing activities

It is necessary to coordinate activities across regions, offices and disciplines as well as activities undertaken with other professionals which might be represented in other areas and form other groupings in other regions.

17.11 Controlling marketing activities

The control of marketing activities can be divided into the control of general activities (corporate development and general marketing) and the control of project marketing.

General marketing activities can be controlled by using forms structured as shown in appendix L developed by GFJ Inc.. An English translation of the activities listed are given in section 17.8.

The main purpose of the control of project marketing activities is to ascertain that project proposals with associated data are up to date. Project marketing data includes the project capital and fee budget, the chance of success of being accepted as a project and applicable calendar dates. This data forms a first level input into the enterprise longer term project budget. The corporate budget example of consulting engineers BKS Property Limited supplied by Puttergill [99] shown in appendix L, figure L.4 refers as an example.

Professional services marketing control should require:

- a session on marketing feedback in the regular office management meetings
- managers/partners/shareholders to review feedback from persons doing marketing on regular, say monthly, basis.

17.11.1 Outcomes and products of the marketing process

The hierarchy of outcomes of the marketing process is outlined in the table below. All the entries represent business artefacts which need to be recorded and managed as required.

Phases Zero, One, Two and Three of the Salford Process Protocol refer to typical outcomes of the marketing process i.e. demonstration of the need, conception of the need, feasibility outline and feasibility study for a typical engineering project. The Salford process protocol is displayed in figures 20.1 and 20.2 and was developed at Salford University in the UK. Refer to Kagioglou et al. [67].

Professional services marketing products and outcomes include:

- Letter proposals
- Proposal reports
- Preliminary design outlines
- Development project master plans
- Briefs from the client
- Project status reports before feasibility and preliminary design phases are entered.

17.12 Professional Services Enterprise Public Relations and Management

Typical questions on public relation management for the professional services enterprise include:

- What is public relations and public relations management?
- Does a professional service enterprise need to look into public relations?
- How do marketing processes and activities tie in with public relations processes and management?
- What are the implications of public relations processes and activities for the form and functional requirements of a kernel business model?

17.12.1 Investigation and forecasting for public relations

Investigation and forecasting activities for public relations include:

- Formulation of issues and problems and setting of goals
- Collect and collate information internal and external to the organisation and do SWOT (Strengths, Weaknesses, Opportunities and Threats) type investigations
- Analyse the information in a two step process focusing on the past and present and then on the future doing the necessary forecasts and estimates
- Decide on a solution: What is to be done? Develop and evaluate alternatives and select a suitable one.

17.12.2 Planning, estimating and budgeting for enterprise public relations

The business policy (vision, mission) provides the strategic and planning framework for the enterprise which is set out in the:

- High level strategy and policies
- Strategic plan and planning framework
- Five year, one year financial plan and quarterly and short term planning estimates and budgets.

Planning, estimating and budgeting activities for public relations include:

- Deciding on actions and approach leading from the investigation (How is it going to be done?)
- Doing a detail investigation and develop detail plans which define actions to be taken.
- Formulating and communicating plans using programming, scheduling and budgeting techniques
- Developing controls for global and detail plans consisting of standards to be applied, measurement and corrective action.
- Developing contingency plans based on risk analysis and risk management principles.

17.12.3 Organising for public relations

The organisation framework for public relations follows the same outline set out in section 16.2.1. It is applied to public relations requirements in this case.

17.12.4 Activities and processes

The procedural framework defines processes, activities and procedures which need to be dealt with under the dynamic aspects of management. These aspects of management i.e. leading, coordinating and controlling are described in section 16.2.2 and need to be applied to public relations management as well.

17.12.5 Leading, coordinating, controlling and communicating public relations activities and processes

Leading, coordinating, controlling and communicating public relations activities and processes follows the approach given in sections 16.2.2 and 16.2.3 applied to public relations management.

17.13 Conclusion and recommendations

With reference to

- the marketing related business objects and processes for a professional services business identified in this chapter
- the type of management reports shown in appendix L
- as the theory and techniques available to construct management systems discussed in chapters 11, 12 and 13

the marketing management function can be well served with effective and efficient marketing systems and marketing management systems.

Chapter 18

Finance, Bookkeeping and Auditing

18.1 Introduction to professional service business accounting

Figure 18.3 contains an overview of the finance, bookkeeping and auditing functions in the professional engineering services enterprise.

An analysis of the data structures, information content and reporting requirements of a typical general ledger accounting system for professional practice management is set out.

Tables 18.1 and 18.2 contain a classified business financial process and object list. The objects and processes identified can be used to build models of the bookkeeping and financial systems of the business.

18.2 Registration and recording processes

The project costing process for a professional services project is based on costed professional time spent on the project as well as business in-house disbursements of goods, materials and resources. Project expenses also arise from goods and services ordered and paid for in cash or via the normal creditor system which are allocated to the project.

Company management and administration activities can also be recorded to administrative projects registered in the project system. As an alternative, special time and expense categories can be defined with the associated reporting functionality included in the time and expense subsystems to manage overheads.

All the registration and recording processes outlined below require suitable database reporting to supply feedback to managers, project leaders and personnel as required.

18.2.1 Project registration

As soon as a project has been identified and its pursuit approved by management it is registered in the project database and all related information on the project recorded. The key to the efficient execution of a project is the commitment required from the project leader which is assigned to the project. Project rate tables for personnel time as well as disbursements which meet the clients specification needs to be set up per project.

18.2.2 Debtor registration

A production project should be allocated to a debtor, which needs to be registered, if it is not active on the database yet. The debtor database should contain a link to the client database or vice versa.

In-house administration projects do not require a debtor.

18.2.3 Creditor registration

Creditors linked to projects as well as other creditors need to be registered in the database with all relevant information as required.

18.2.4 Time keeping and recording

Time keeping is typically done on time sheets. The time sheet forms can be paper based or input via computer programs using suitable system user interface or web page based forms.

Table 18.1: Finance, accounting and bookkeeping processes and objects

Process	Activity	Related accounting objects	Notes
Financial calendar setup	Set up financial calendar with dates and periods to be used for financial year	Calendar	More than one financial year might be open at a time
Time recording	Time sheet entry Time sheet processing and control	Time sheets Time reports	
In-house Disbursement recording	Telecommunications Courier and postage Document copying and binding Travel expenses Accommodation Subsistence Entertainment Laboratory Computing Consumables Specialised services	Project service requests In-house Disbursement reports	Goods and services sourced in house on a cost basis to support production projects. Expenses/costs recorded to be accounted for in project expenses. Can be recovered from projects and invoiced.
Project ordering and creditors	Professional services bought Telecommunications Courier and postage Document copying and binding Project travel expenses Project accommodation Project subsistence Project entertainment Laboratory services Computing Specialised services	Project expense orders Project disbursement reports	Goods and services ordered to support production projects. Expenses/costs recorded to be accounted for in project expenses. Can be recovered from projects and invoiced.
Business ordering and creditors	Professional services bought Consumables Telecommunications Courier and postage Document copying and binding Travel Accommodation Subsistence Entertainment Computing Specialised services	Orders Creditor reports	Goods and services ordered to support business processes. Recorded as overheads.
Professional time costing		Personnel register Rate tables	Produce classified time rate tables.
Project registration		Project register Project list	
Client registration		Client register Client list	Link to debtor implied.
Personnel creditor registration		Personnel register Personnel creditor list	Can be used to administer contract labour expenses.
Personnel debtor registration		Personnel register Personnel debtor list	
Petty cash administration	Administer cash Record expenses		Expenses can be project or business (overhead) related.
Project budgeting	Project time budgeting Project income budgeting Project expense budgeting	Project budgets	

Table 18.2: Finance, accounting and bookkeeping processes and objects (continued)

Process	Activity	Related accounting objects	Notes
Invoicing	Project invoices and tax (VAT) invoices Project credit notes and tax (VAT) credit notes Personnel invoices Personnel credit notes Other invoices Write up expenses to quoted prices Write off expenses exceeding quoted prices	Invoicing budget Invoices	
Asset recording	Record new assets Write off old assets Depreciation accounting	Asset register Asset report	
Remuneration	Salary determination Salary adjustments Periodic payment processing	Salary scales Salary payment report	
Financial reporting	Creditors reporting Debtors reporting Invoicing and sales reporting Income / expense accounting transactions	Creditors journal Debtors journal Sales journal Cash journal	
Project reporting	Project status and expense reporting as required	Various project reports	
General ledger processing	General ledger account structure design General ledger account maintenance General ledger journals	General ledger General ledger journal	
Banking	Bank account reconciliation	Bank statement Bank reconciliation report	
Auditing	Payroll and personnel cycle audit Acquisition and payment cycle audit Consumables inventory and storage cycle audit Capital acquisition and repayment cycle audit Cash balance auditing	Audit plan and programme Audit working papers Audit report	Audit only required by law for companies.

The selection of daily, weekly or monthly cycles for time sheets is typically dictated by the time management cycle as well as the invoicing cycle of the business. With the widespread use of computer workstations and communication networks in businesses, the completion of a daily time sheet should not be a problem for the disciplined worker. This can also interact with the diary recording requirement of a typical professional.

An example of a basic time sheet is shown in figure 18.1.

18.2.5 Disbursement recording and costing rates

To record in-house disbursements such as travel, subsistence, copying, document production, postage and courier, contract labour, laboratory expenses and other costs, rate tables are required. Some rate tables are prescribed by clients and other are computed by in-house costing clerks and managers.

Project and overhead disbursements are recorded as they are made. A suitable recording facility which ensures disciplined data capture linked to the project reference or overhead cost code is required.

[illegible]

Figure 18.1: *Basic time sheet example*

18.3 Work in process

Work in process (WIP) is a very important key asset of a professional services organisation. It is the equivalent of the finished goods inventory of a manufacturing business and needs to be managed with care. WIP forms the basis of invoice generation for both time and expense base invoices (refer to section 18.5) as well as fee based (fixed price) invoices.

18.3.1 Professional time Work in Process

Professional time work in process represents all the project time expended on a project which has not been catered for in an invoice.

18.3.2 Disbursement Work in Process

Disbursement work in process represents all the project related disbursements i.e. in-house as well as project creditor related which have not been taken up in an invoice.

18.3.3 Work in process management

A number of possible actions can be applied to a work in process entry.

- It can be *included* in a time and expense invoice as such
- It can be *allocated* to an invoice item in a fee based (fixed price) invoice
- It can be *written off* and not included in an invoice
- It can be *written up* to correspond to an invoice item in a fee based (fixed price) invoice

Whether formal accounting of WIP is required is an open issue which is discussed in section 18.7.2.12.

18.4 Professional Services Invoices

The professional services invoice is a key document in the administration of the monetary income stream of a professional services organisation. This section defines an invoice and describes the format of the document as well as the procedure for generating an invoice as well as the links to related datasets feeding data into the invoicing process as well as the datasets receiving the invoice data.

Refer to table 18.3 for a specification of a professional services invoice.

18.4.1 Definition of an invoice

An invoice is a legal document issued by a business enterprise to a client/customer to indicate the extent of money owed for services and goods rendered to the client. It is a 'bill written by a seller of goods or services and submitted to the purchaser'.

18.4.2 Management of the invoicing cycle

The invoicing process per project is typically triggered by an entry in the project or office cash flow budget. The budgets are reviewed on a monthly basis to decide which projects are at a stage at which the client can be invoiced.

Some organisations use pro-forma invoicing to issue draft invoices which are reviewed by clients before the final invoice is edited and issued to the client.

Reference to the enterprise document management system to supply copies of invoices of goods and services bought as part of project disbursements might be required. These documents typically need to be added as supporting documentation to invoices.

18.4.3 Responsibility for issuing of invoices

The project leader or engineer or any personnel member higher up in the enterprise hierarchy can be responsible for issuing project invoices and can delegate the activity to suitable administration personnel available.

A typical professional services invoice contains data reflecting the information set out in table 18.3 and can be structured as shown in figure 18.2.

It is important to ensure that any requirements/specifications of a client for invoices issued to the client are adhered to. This should prevent time absorbing interaction with the client where modification of invoices and issuing of credit notes are required. The level of detail of an invoice as well as supporting documentation required by a client needs to be determined at project startup to ensure efficient invoicing, processing by the client of the invoice and prompt payment by the client.

18.5 Professional Service Invoice Specification

In general an invoice can contain fixed price line items as well as computed quantity/unit price line items. The Work in process (WIP) professional time and labour as well as expenses/disbursement entries on record per project needs to be reconciled with each invoice entry to allow control of the WIP.

The source data referred to for computed line items will always be reflected in the WIP labour or disbursement record. If computed line items are added to an invoice where WIP is not referenced, WIP write up entries for the project need to be generated.

A fixed price line item can contain a calculation indicating how the amount is derived from a set fee calculation, based on a client requirement or client brief document reference.

Fixed price line items placed in an invoice need to be linked to the applicable labour and disbursement WIP entries which apply to the specific item to ensure integrity of the WIP record for the project.

Table 18.4 lists the information hierarchies which the invoicing specification and generation process links to.

A sample tax invoice display screen is shown in figure 18.2.

The invoicing data is transferred to the accounting dataset when the invoice is generated. It is also then reflected in the debtor database.

Table 18.3: Professional service invoice specification

Invoice date
Data of the enterprise issuing the invoice
Data of the client to which the invoice is addressed
A heading with data on the project/activity to which the invoice relates
Fixed price specified invoice line items
Computed quantity/ unit price invoice line items
Value added tax (VAT) percentage and amount
Currency in which monetary amounts are stated
Special terms and conditions which apply to the invoice
and or services and goods referred to on the invoice

Table 18.4: Information hierarchies which link to invoice specification and generation

Client dataset
Cash flow budget dataset
Fee calculation dataset
Professional labour rates dataset
Work in process
Accounting general ledger
Value added tax tables
Debtors

Stadium Construction & Hardware Pty Ltd
17 Australia Street
Homebush
Sydney, 2058

Tax Invoice

ABN	Date	Tax Invoice #
14 003 348 730	12/03/2004	33

Tax Invoice To
Zacharias J. Juppansen
Block C, Deve Street
Cyberland
Sydney, NSW 2085

P.O. No.	Terms	Project
	Due on receipt	Clients account

Description	Qty	Rate	TAX AMT	Amount
	2.6	25.00	5.91	65.00
	5	25.00	11.36	125.00

Quicken Additional terms, conditions and/or fees may apply for certain features and optional services. II PAUSE << REWIND X QUIT

Done Internet

Figure 18.2: Tax invoice Example Screen

18.5.1 Value Added Tax (VAT)

South African tax legislation requires that business enterprises registered as tax vendors need to issue 'tax invoices' which comply with the requirements of the South African Revenue Service (SARS). Publications such as Huxam and Haupt [63] provide information on business VAT registration requirements and the business VAT process.

18.5.2 Accounting records and reporting on invoicing

As soon as an invoice is generated it is reflected in the applicable general ledger accounts.

The Sales Journal report, which is drawn from the general ledger accounts, supplies management information on the invoicing activity for a selected period.

The detail general ledger income accounts required for the sales activities are listed in table 18.5.

Table 18.5: General Ledger Basic Sales Accounts

PROFESSIONAL FEES
ACCRUED PROFESSIONAL FEES
DISBURSEMENTS RECOVERED
ACCRUED DISBURSEMENTS RECOVERED
OTHER INCOME

The detail ledger liability accounts required for VAT accounting are listed in table 18.6.

Table 18.6: General Ledger VAT Accounts

VAT Account	SARS form 201A line No.	VAT item
VAT OUT Supplies Goods/Service standard rate	1	
VAT OUT Supplies Accommodation	9	
VAT OUT Adjustments Change in use	11	
VAT OUT Adjustment Other	12	
VAT IN Goods/Services	14	
VAT IN Capital Goods/Services	15	
VAT IN Adjustments Change in use	16	
VAT IN Adjustments Bad Debts	17	
VAT IN Adjustments Other	18	
VAT Return Clearing		
Accrued VAT		

18.5.3 The debtor cycle

As soon as an invoice has been issued and delivered the client becomes a debtor of the business and is liable for payment of the invoice. Debtor control follows from monthly debtor listings with debtor ageing information which can be followed up by administrators and management. Up-front agreements with clients on payment terms should ensure prompt settlement of invoice payments due.

18.5.4 Credit notes and cancelling of invoices

If an invoice needs to be withdrawn or amended it is good practice to issue a credit note for the whole invoice. A new invoice is then generated as required and the previous invoice is marked as deleted. The WIP reversal, debtor system update, VAT reversal requirements imply that a consistent and complete invoice reversal process which is the inverse of the invoice generation process is available as part of the functionality of an enterprise financial management system.

18.5.5 Internal invoicing to personnel

A formal invoicing system of recovering expenses such as purchases made on behalf of personnel by the enterprise or supply of goods and or services by the enterprise directly to personnel, can be administered as part of the normal project invoicing operation or as a distinct invoicing operation. This indicates the possible requirement of an invoicing operation information hierarchy for the enterprise.

Table 18.7: Personnel and payroll general ledger accounts

Salary
Overtime paid
Leave paid out
Entertainment allowance
Car Allowance
Travel Reimbursed
Home and Office Rental
Computer Allowance
Annual Bonus Payments
Share and Loan Account Insurance
Spouse Insurance
Pension Contribution
Medical Aid Contribution
Unemployment Insurance Fund (UIF) Company Contribution
Workman's Compensation Contribution
Income Insurance
Other Company Contributions
Provision for leave payment
Provision for bonuses

18.6 Personnel remuneration and payroll processing

Personnel payroll processing is discussed in section 19.3. As far as bookkeeping is concerned a set of general ledger accounts are set up which link with the payroll processing system to record the financial transactions involved. Sections for shareholders, technical personnel as well as administrative personnel need to be maintained to support typical reporting requirements.

Table 18.7 lists general ledger accounts associated with payroll and personnel financial transactions.

18.7 Bookkeeping

Bookkeeping refers to the accounting processes taking place around the general ledger of accounts, cash books, creditors and debtors. Asset register processing can also be classified under bookkeeping.

18.7.1 Accounting general ledger structuring and format

The general ledger contains all the accounts accessed in an accounting system as well as a series of control accounts which are used as links to any subsystems linked to the general ledger.

18.7.2 The role and use of the general ledger in accounting

As a rule the general ledger is underutilised in accounting. The level of detail recorded in the general ledger account structure dictates the level of detail to which reporting can be done for bookkeeping and accounting.

With the advent of computer bookkeeping applications the number of accounts in the general ledger does not impact on the workload of an accountant or auditor.

18.7.2.1 Structuring of the general ledger

Table 18.10 and table 18.11 shows an overview of an approach which can be adopted to structure a general ledger for a professional services enterprise. This general ledger structure is based on development work by Robertson [103] which describes an integrated practice management and accounting system.

Ill conceived general ledger structures like the alphabetical list of accounts provided by the audit profession with a view to income tax requirements should be avoided. The 'Standardised Accounting System' (South African Association of Consulting Engineers [113]) defined by the South African Association for Consulting Engineers (SAACE) is an example of a poorly devised general ledger structure which should not be used. If accounting data conforming to this ill conceived logic is required accounting reports can be set up to supply data in this format.

Table 18.8: General ledger matrix model

Account Identifier	Balance period 1	...	Balance period n	Budget period 1	...	Budget period n
identifier
identifier
Subtotal 1:	<i>Subtotal 1 balance period 1</i>	...	<i>Subtotal 1 balance period n</i>	<i>Subtotal 1 budget period 1</i>	...	<i>Subtotal 1 budget period n</i>
identifier
identifier
Subtotal 2:	<i>Subtotal 2 balance period 1</i>	...	<i>Subtotal 2 balance period n</i>	<i>Subtotal 2 budget period 1</i>	...	<i>Subtotal 2 budget period n</i>
Total:	<i>Total balance period 1 = 0</i>	...	<i>Total balance period $n = 0$</i>	<i>Total budget period 1 = 0</i>	...	<i>Total budget period $n = 0$</i>

Table 18.9: Debit and credit transaction logic: effect on account balances

Balance sheet accounts	Debit	Credit
Assets	Increase (+)	Decrease (−)
Liabilities	Decrease (−)	Increase (+)
Owner's equity	Decrease (−)	Increase (+)
Income statement accounts	Debit	Credit
Income	Decrease (−)	Increase (+)
Expense	Increase (+)	Decrease (−)

18.7.2.2 Abstract mathematical model of an accounting system

An abstract mathematical model of general ledger based accounting systems is given in [86]. The general ledger can be mathematically represented as a matrix. The matrix model consists of identified rows (accounts) with a selected number of columns containing structured account balances for i.e. budget and actual vales per period. Transactions modify any matrix entry and column totals need to remain zero if a sign convention of debit (+) and credit (−) is adopted. This concept is depicted in table 18.8.

18.7.2.3 General ledger editing and updating

The implementation of general ledger data structure in commercial accounting packages is typically that of a linked list. Accounts can be added and deleted. Accounts which contain transaction history typically need to be retained until the end of a financial period after which they can be deleted without any future referencing required.

18.7.2.4 Debit and credit

Double entry accounting transactions follow the logic outlined in table 18.9.

18.7.2.5 Cash books

Cash books record cash, cheque or electronic payment transactions at any point in the business where these take place. Cash books systems are in the form of subsystems which link to the accounting system via cash control and bank accounts. Bank reconciliation is required to ensure that the transactions recorded in the bank account by the bank are in synchronisation with the transactions recorded in the business.

18.7.2.6 Creditor system

The creditor (accounts payable) system is a subsystem of the accounting system dealing with creditor transactions. Creditor transactions originate when goods or services are ordered from and supplied by suppliers on credit to the business.

All detail information or creditors and transactions with creditors are maintained here.

Special control accounts can be linked to the creditor subsystem which contain detail information about all creditor transactions and can produce required journal listings as well as analysis and creditor payment, discount and ageing reports.

A separate personnel creditor system can be set up to administer payments to be made to personnel for goods and services supplied to the business.

18.7.2.7 Debtor system

The debtor (accounts receivable) system is a subsystem of the accounting system which records the status of invoices issued to clients. When payments are made for invoices issued balancing debt reduction and cash balance increasing transactions are recorded.

Special control accounts are linked to the debtor subsystem which contain detail information about all debtor transactions and can produce the required journal listings as well as analysis and ageing reports.

A separate personnel debtor system can be set up to administer payments to be received from personnel for goods supplied and services rendered by the business to the personnel.

18.7.2.8 The trial balance

The trial balance is a standard report listing all accounts in logical sequence with associated balances at a given point in time to check that the system is in balance.

18.7.2.9 General ledger journal transactions

General ledger journal transactions are used to adjust and account balance as required. The transaction must be defined to keep the accounting system in balance.

A general ledger journal transaction is defined and then posted to the general ledger in the accounting database.

18.7.2.10 General ledger based accounting reporting

The general ledger structure should be able to supply the base data for a number of reporting functions such as the balance sheet report and income statement reporting in various formats. Tree structured reports to meet any management, tax or auditing requirement can be defined as long as the granularity of the underlying account structure is good enough to accommodate the report data reference requirements. Refer to chapter 12 for the techniques for defining such reports.

18.7.2.11 Financial period end processing

Accounting systems have financial period end processing functionality which has the purpose of transferring income/expense totals to accrued balance sheet entries at financial year end.

Calendar settings for monthly or yearly control of transaction entry are also set during these operations.

These operations are not strictly required in a system and can be done using general ledger journals. The finite number of columns available for structuring general ledger transaction per accounting period typically dictates the availability of this kind of functionality.

18.7.2.12 Accounting for WIP and costing data in the general ledger

It is possible to carry WIP and costing data in the general ledger. The costing or WIP accounts (or control accounts used as a link to any costing or WIP system) should however be grouped logically so that the balances of these accounts can be identified easily for inclusion or exclusion from structured financial reports.

Reporting structures, based on the reconciliation of costing and accounting data, can be used to determine cost rates e.g. the popular hourly technical personnel cost rate per 100 units of salary package (20c/R100, 15c/R100, ...) on an historical basis. The level of overhead costs can e.g. be determined in this way and managed accordingly.

18.7.2.13 Activity-Based Costing and Management

Glad and Becker [42] provides an overview of activity-based costing and management. The shortcomings of the traditional model of cost management are described. The use of a properly structured project system linked to a general ledger which contains a suitable level of detail can support any requirement of this costing model.

18.7.3 Accounting system model

A data model for an accounting system as a database can be based on the structure shown in tables 18.10 and 18.11.

Database transactions to process accounting activities outlined above are required.

18.7.4 Accounting software implementation

Standard commercial accounting system software can be used to implement the general ledger account structure shown here. Financial and costing accounting transactions can be generated by project, debtor, creditor, personnel salary, asset and facility management, laboratory and other subsystems as required and imported into the accounting system on a regular basis. Geographically decentralised systems with suitable network or other data links can also be set up, as required.

Table 18.10: Overview of the typical structure of a general ledger - Assets and Liabilities

<i>Level</i>	<i>Description: Level 1</i>	<i>Description: Level 2</i>	<i>Description: Level 3</i>	<i>Description: Level 4</i>
A	Balance sheet accounts			
1.	Assets			
1.1		Enterprise assets		
1.1.1			Fixed assets	
1.1.2			Current assets	
1.2		Office assets		
1.2.1			Fixed assets	
1.2.2.1			Instalment sale assets	
1.2.2.2				Leased assets
1.2.2.3				Rental assets
1.2.2.4				Accumulated depreciation
1.2.3			Current assets	
1.3		Departmental/Divisional assets		
2	Owners Equity			
2.1		Capital		
2.2		Loan accounts		
2.3		Retained earnings		
3.	Liabilities			
3.1		Long term liabilities		
3.2		Current liabilities		
4	Balancing/Clearing accounts			

18.8 Processing orders for materials, goods and services

Orders for material, goods and services for project work which are not supplied and stocked in-house in the business can be obtained on a controlled order basis and paid for in cash or via the creditor system. The approved project budget will typically indicate requirements of this nature.

Table 18.11: Overview of the typical structure of a general ledger - Income / Expenses

B	Income/ Expense accounts			
5.	Income			
5.1		Enterprise Income		
5.2		Departmental/ Divisional Income		
5.2.1			Recoverable expenses	
5.3		Office Income		
6.	Expenses			
6.1		Enterprise Expenses		
6.1.1			Cost of sales - enterprise	
6.1.2			Administration	
6.1.3			Personnel recruitment and benefits	
6.1.4			Donations	
6.1.5			Interest paid	
6.1.6			Professional liability insurance	
6.1.7			Discount received	
6.1.8			Dividends paid	
6.2		Departmental/Divisional Expenses		
6.2.1			Cost of sales - department	
6.2.2			Project production expenses	
6.2.3			Promotional expenses	
6.2.4			Personnel development	
6.2.5			Administrative expenses	
6.3		Office Expenses		
6.3.1			Cost of sales - office	
6.3.1.1				Salaries
6.3.2			Administration	
6.3.3			Depreciation	
6.3.4			Repair and maintenance	
6.3.5			Rental of equipment	
6.3.6			Insurance	
6.3.7			Losses and write offs	
6.3.8			Consumables and sundries	
6.3.9			Rates and rentals	
6.3.10			Profit sharing and bonuses	

18.9 Professional practice finance

Finance for service providing professional practices is typically obtained from the proprietor, partners or company directors as shareholder loans.

Short term finance can also be obtained in bank overdraft form as required.

18.10 Auditing

Auditing processes are required by law for companies according to the Companies Act. The main purpose of auditing is to ensure the quality of financial reporting in the business. Accounting data is audited to ensure that the recorded and reported information accurately reflects the economic events that occurred during the accounting period.

An audit plan and programme is drawn up and the auditors draw up audit working papers as documents and systems are checked.

Audits of the following business cycles and balances are done:

- Payroll and personnel
- Acquisition and payment
- Consumables inventory and storage
- Capital acquisition and repayment
- Cash balances

Texts such as Arens and Loebbecke [5] and Heymans [55] provide an overview of the auditing process and requirements as well as detail information on procedures to be followed during a business financial audit.

18.11 Conclusion and recommendations

The review of the record keeping for costing and finance, bookkeeping and auditing functions of the professional services enterprise in this chapter indicates that the systems can be modelled and structured using basic mathematical constructs and database concepts such as those discussed in chapters 4 and 7.

Commercial accounting software systems can be linked to other business systems to provide generalised management reporting functionality structured as described in chapter 12. These business systems can be database driven.

Engineering Office Management System Diagram

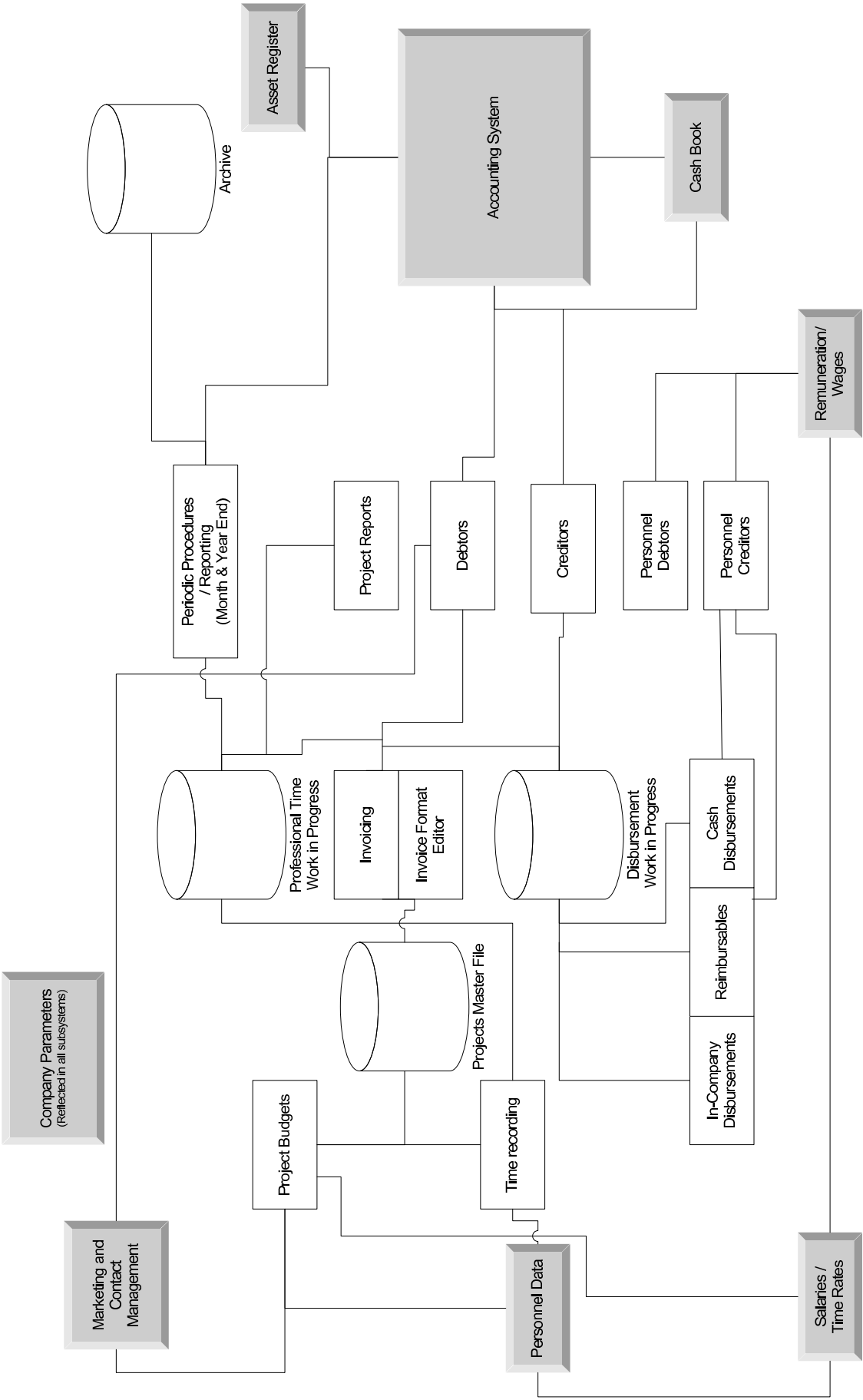


Figure 18.3: Engineering Services Enterprise Systems

Chapter 19

Personnel Management

Personnel management is also termed human resources management.

Models for personnel management consist of business objects and activities and processes for personnel management for a professional services business.

19.1 Business objects for personnel management

A listing and classification of personnel related business objects is given below.

19.1.1 High level logical personnel management objects

High level logical personnel management objects are listed below. The approach given here is being adopted in the OnePurdue initiative to develop new Enterprise Resource Systems for Purdue University by integrating mission-critical enterprise data, information and business processes. Refer to Purdue University - OnePurdue Project [98].

- Persons: Objects that hold positions within an organisational structure
- Jobs: General classifications of groupings for sets of tasks of functions an employee is required to perform
- Positions: These are instances of jobs and can be one-one for single employee filled positions or grouped for multiple employee filled positions
- Organisational Units: Business units which provide the structure to which positions are linked
- Cost centres are maintained in financial accounting and typically default to Organisational Units

19.1.2 Personnel management objects

Other objects which play a role in personnel management are listed below.

- Calendar with working days
- Person or personnel member
- Jobs or posts with job descriptions and staffing plans
- Organisational Units – as defined above - these can be logical entities such as offices, technical departments and administrative service departments
- Organisational structure – a graph defining logical links between jobs and organisational units
- Salary and remuneration scales
- Costing rates and tables typically linked to salary scales
- Personnel records and forms
- Training and course material

- Office infrastructure (refer to Facility management in chapter 21)
- Personnel tax schedules
- Personnel deduction schedules for e.g. pension, insurance etc.
- External compliance documentation e.g. labour law documentation
- Internal compliance documentation e.g. company strategy and policy documents
- Labour unions and personnel organisations

19.2 Personnel management processes

Personnel management processes identified by Gerber et al. [41] include:

- Human resources strategy development and planning
- Structuring of organisation
- Job description development
- Recruiting for personnel
- Selecting personnel
- Placing and induction
- Day to day planning, organising and controlling of job related work
- Career guidance and management
- Intrinsic personal motivation
- Performance appraisal
- Remuneration
- Fringe benefit goods and services allocation
- Quality of work life assessment and management
- Health and safety management
- Leadership development
- Group formation and organisational development
- Labour relations management
- Training
- Management development
- Management of the technology-human interface.

19.3 Payroll management systems

Payroll management systems are typically customised for the requirements of a specific country.

Systems which are available in South Africa include Pastel Payroll Software Softline Systems [111] and VIP Payroll Software Softline VIP [112].

19.4 Personnel Debtors

A personnel debtors system is used to record goods or services provided by a business to its personnel and manage the payment for the goods or services. Typical applications are company funded housing, transport and other benefits.

19.5 Personnel Creditors

A personnel creditor system is used to record goods or services supplied to a business by its personnel. Contract hourly paid labour can be managed in this way.

19.6 Conclusions and recommendations

The review of the personnel function of the professional services enterprise in this chapter indicates that personnel systems can be modelled and structured using basic mathematical and database concepts discussed in chapters 4 and 7.

Reported case studies of academic institutions, where the personnel function is supported by sophisticated workflow and management support and reporting systems, can be used as input for to develop systems for the professional services business.

Commercial personnel payroll software systems can be linked to other business systems to provide generalised management reporting functionality structured as described in chapter 12. These business systems can be database driven.

Chapter 20

Production

20.1 Introduction

The production process in the professional service business such as consulting engineering and architecture revolves around the project. The project is viewed as the basic unit of work. In this chapter aspects of project and project management including terminology used, project objects as well as project processes which can be identified are discussed.

See Pieter Smit [110] for more detail.

20.2 Projects and project management

The project is the basic unit of work in a professional service organisation. The project has a given start time and end time, is linked to a specific engineering or other product, infrastructure element or service to be manufactured, constructed or developed for the benefit of the owner or project client as well as other stakeholders in the project.

On large projects clients can also brief professional practices to manage the procurement cycle for equipment to be built into constructed facilities as part of a project.

The complexity of project management in the professional practice increases with the advent of network linked distributed environments in which work is executed.

20.3 Project management terminology

The discipline of project management abounds with terms which can be defined as required and related to object hierarchies. Terms like programme, project, activity and work package relate to different units in which project activities and groups of projects are grouped. A process can also be viewed as a special type of project. Table 20.1 lists a number of these terms. Some of the definitions were sourced from Burke [21].

20.4 Production management business objects

Production management business objects can be referenced to the theoretical approach outlined in Chapter 11.

The basic production management business objects are:

- persons or groups of persons
- tasks
- tools used to perform tasks
- units of data operated on by persons, tasks and tools

Tasks (activities) can take the form of planning, design, specification and documentation activities grouped together to form a logical project.

Other related objects which are typical instances of data or units of data (datasets) include project related:

Table 20.1: Project management concepts and terms

<i>Project Management Concept</i>	<i>Description</i>
Project	A basic unit of work that has a given start time and end time.
Activity	A task, job or operation to be performed to complete a work package or project
Process	A series of activities which are performed to achieve a business goal
Work package	A group of project activities which can be performed in a given location by a given set of project resources and which can be estimated, planned and assigned to a responsible person or department for completion
Work breakdown structure	A work breakdown structure (WBS) subdivides the scope of work into manageable work packages
Resource allocation	The linking of a resource (manpower, tool, materials, goods or services) to an activity or work package
Earned value analysis	The estimating, recording and reporting of a parameter or set of parameters based on the earned value structure of the project to monitor project progress
Configuration control	The scope of work of the project is stated and updated as required
Project life cycle	Phases that a project passes through. Typical life cycle phases identified are concept and initialisation, design and development, implementation and construction and commissioning and handover.
Gantt chart	Display of project activity logic on a time scale
Critical path method (CPM)	The identification of the project activities on the critical path i.e. activities whose execution timing controls the total project time
Program Evaluation and Review Technique (PERT)	PERT is a logical network based method to analyse the tasks involved in completing a given project and to compute the minimum time needed to complete the total project.
Project management: Project Scope - Project Time - Project Cost - Project Quality - Project Human resources and teams - Project Communications - Project Risk - Project Procurement and Logistics -	Project Scope - The process required to ensure that a project contains all the work required to complete it successfully Project Time - Select project calendar and identify time periods form project Project Cost - Total of all costs expended on project Project Quality - The process to ensure that the project will satisfy the needs for which it was undertaken Project Human resources and teams - Staffing and management of project personnel Project Communications - The process required to ensure timely handling of project information Project Risk - Evaluation and management of uncertainties associated with a project Project Procurement and Logistics - Supply of resources and storage of material for project
Project organisation structures	Any structure linking tasks, persons, and tools for management purposes
Project costing and estimating	
Resource planning: scheduling and levelling	Resources required to execute project activities are allocated and activities and or resources adjusted to solve undersupply and oversupply situations
Project accounting, payment certification	Reviewing project payment claims by contractors and suppliers and certifying invoices for payment by the client

- project calendar
- plans
- budgets
- communication objects such as letters, proposals, internal reports and external reports
- bills of quantities
- specification documents
- tender documents
- contract award documents
- contract administration documents

- photographs, videos and other image and voice recordings
- construction payment claims, certificates and invoices
- completion reports
- professional fee invoices

The project and its component activities can be modelled as a business object linked to a series of defined hierarchies for business reporting purposes.

20.5 Project management processes

Typical project processes which can be identified are:

- Project identification and conceptualisation
- Coordinating activities with client and other parties
- Proposal development
- Planning and budgeting
- Design
- Specification and documentation
- Tender administration
- Contract administration

Project management processes which are designed to steer and control the project processes are:

- Project team staffing and organising
- Production budgeting and time scheduling
- Project scoping to determine output required
- Identification and sourcing of project tools and resources required
- Production planning
- Production scheduling and control
- Record keeping of activities
- Client invoicing

Figures 20.1 and 20.2 show the concept and structure of a process protocol model, developed at Salford University in the UK and funded by Physical Sciences Research Council (EPSRC), referred to in Kagioglou et al. [67]. This model provides a general design and construction process protocol.

20.6 Functionality required of a project management system for professional services

A project management system which is suitable for the professional practice environment needs to support the following functionality.

- project status setting i.e. proposal, time and expense, fee basis, administrative, completed and other as required
- link to client database
- project logic structure model i.e. activities and sub projects

- project budgeting per activity linked to personnel rate and disbursement datasets
- project professional time and disbursement recording
- project creditor management
- currency conversion for foreign projects
- link to time and expense basis as well as fixed price invoicing
- work in process reporting and management

20.7 Software implementation

Commercial software systems which contain implementations of project management for professional service businesses are discussed in chapter 26.

20.8 Time Management

Time management plays a vital role in the execution of projects. Priorities are typically assigned to project tasks and time estimates made to ensure that programs drawn up can be completed on time. Resource scheduling based on job shop principles is required to ensure productive use of project resources.

Barlow [12, Section 11 - Time utilisation] contains guidelines on time management for the professional engineer.

20.9 Projects for administration management

Non-terminating projects can be used to manage and carry management information for corporate ‘overhead’ enterprise activities. Typical projects which can be defined include strategic planning, regional management, office management, marketing management and other.

20.10 Conclusions and recommendations

The review of the production function of the professional services enterprise in this chapter indicates that production systems can be modelled and structured using basic concepts discussed in chapters 4 and 7.

The production function is one where the Engineering process model approach described in chapter 13 can be adopted with great success. Additional attributes can be added to the tasks, person, datasets and tool objects to model functionality required for production management. Links to other data contained in business databases, such as the personnel and project register databases, can be defined to support system functionality required.

The present implementation of the Engineering process model which uses a step schedule needs to be extended to make time-based, calendar-linked scheduling of tasks possible.

Commercial project management software systems can be linked to other database driven business systems to provide generalised management reporting functionality structured as described in chapter 12.



PROCESS protocol

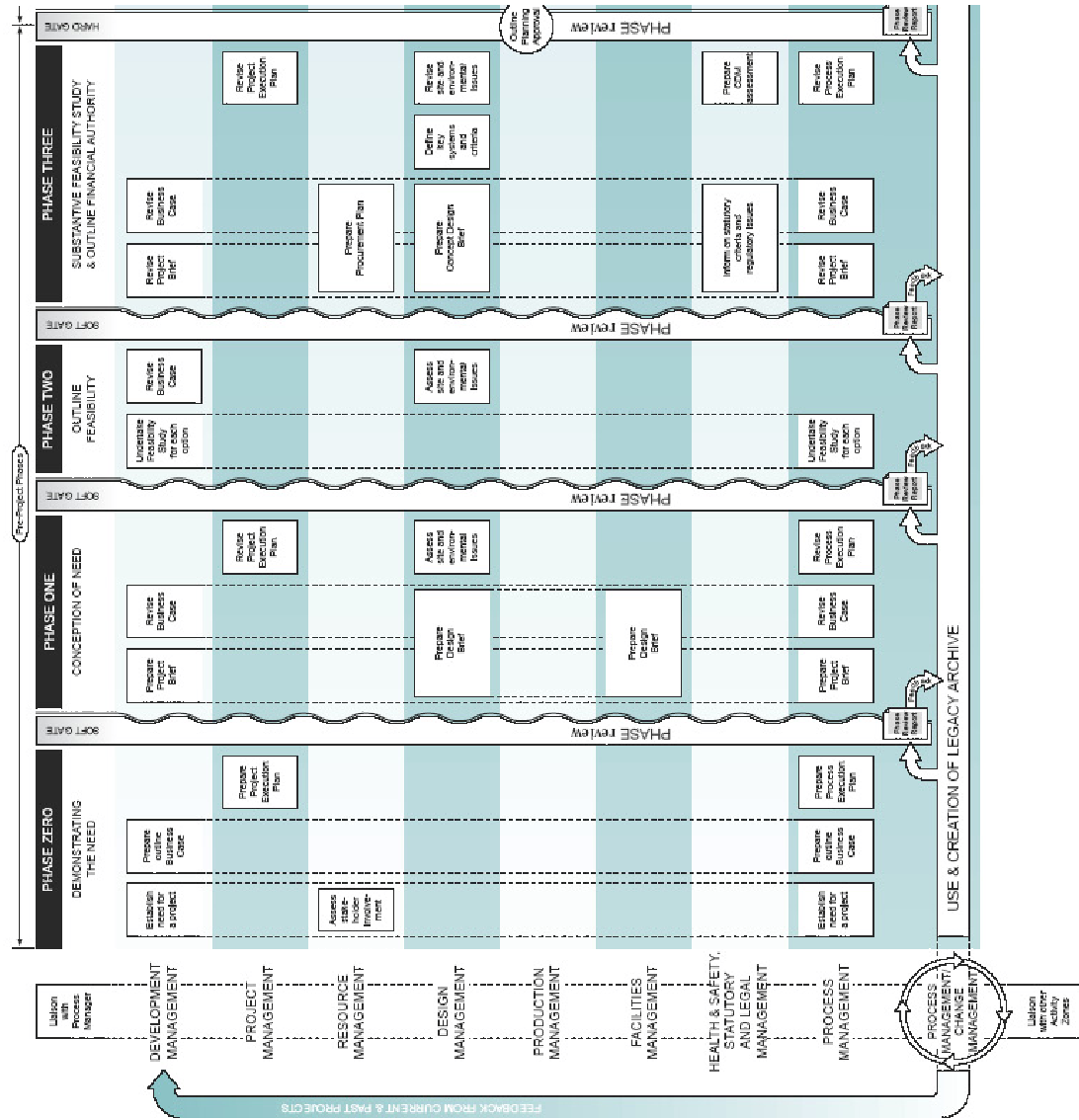
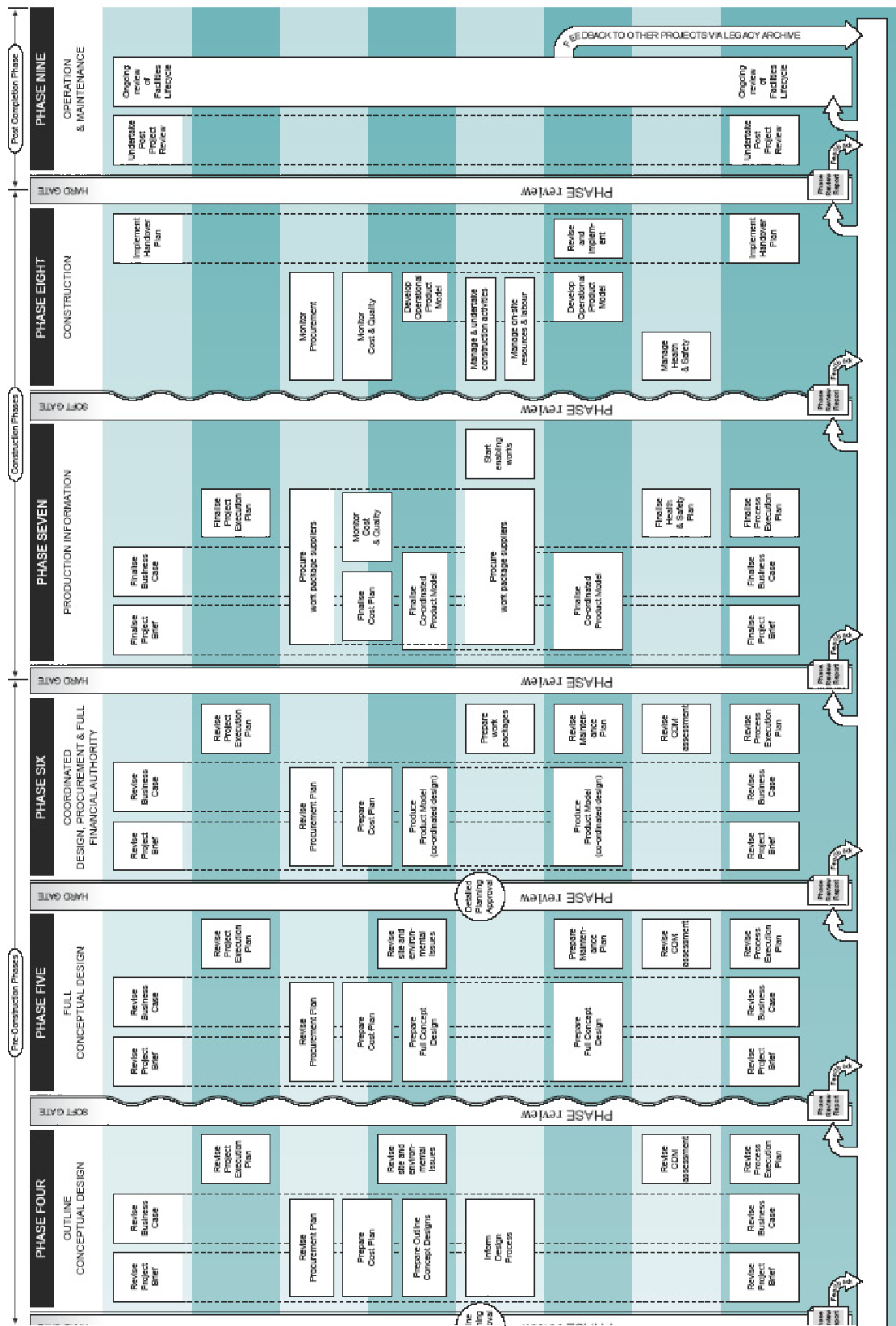


Figure 20.1: Salford Process Protocol - Part A



PROCESS protocol

EPFRC, IMI, Genetix Design & Construction Process Protocol

University of Salford

Alfred McAlpine Construction

Engineering Technology

BAA plc

B.T.

EDM Architects

Walsman Partnership

Soulton & Paul Ltd.

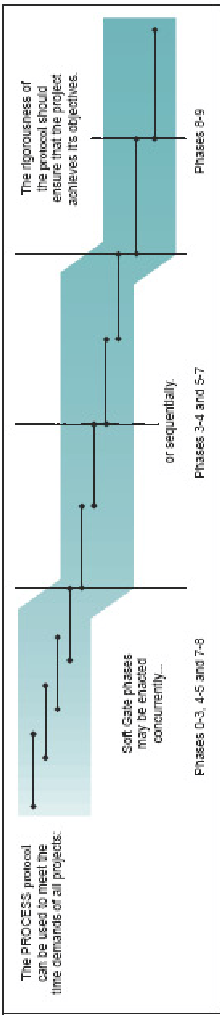


Figure 20.2: Salford Process Protocol - Part B

Chapter 21

Facilities and Document Management

21.1 Introduction to Facilities Management for the Engineering Services Business

Facilities management in the engineering services business environment focuses on the management of physical artefacts required for business processes and activities.

The service business can be the owners or occupiers of building facilities and other infrastructure and should place emphasis on the medium term and long term aspects and processes involved in the optimal utilisation of these assets.

Private sector property owners require market related return on the capital invested in the property that they own and typically manage the property with this goal in mind. In the public sector the management of requirements (benefits) and containing of costs relating to facilities in use, is the approach adopted.

To support the better utilisation of infrastructure Facilities Management (FM) as a management discipline implementing knowledge and procedures from a number of professions has been developing.

The management of artefacts which are data and information related are discussed under Knowledge Management in chapter 22.

This chapter uses material developed in articles by Strasheim et al. [118] and Strasheim [117] as well as an undergraduate project by Rooseboom [105].

21.2 Models for facility management

Models for logistics management consist of business objects and activities and processes for logistics management for a professional services business.

21.3 Business objects for facility management

Selected business objects required for facility management processes are discussed below.

21.3.1 Definition of business artefacts and business objects

Business artefacts are defined as all inanimate physical and abstract entities/objects which are used or referenced in the ongoing business processes of any given enterprise.

Artefacts can be logically classified as physical infrastructure, energy and water supply, waste removal, physical consumable stock items, portable business equipment, tools and computer systems, procedure manuals, books, documents, computer software and intellectual property. Refer to the Matter Energy Information (MEI) technology classification hierarchy developed by Rias van Wyk [123].

Facilities typically contain a number of subsystems which need engineering expertise in the developmental, operational, service and refurbishment phases of the service life of a facility. The office building and internal objects and subsystems listed below have differing service and refurbishment cycles ranging from days to 30 to 50 years.

1. Furniture and desktop equipment
2. Telecommunication cabling and equipment

3. Security and alarm systems
4. Lighting
5. Space division and utilisation partitions
6. Solid waste removal and recycling
7. Air conditioning and ventilation ducts, heating and cooling systems
8. Water supply, waste water removal, irrigation and storm water pipe networks
9. Structural components of the building

21.4 Management disciplines which relate to business object facility management

The International Facility Management Association (IFMA) IFMA [64] lists the functions of facility management shown in table 21.1.

Table 21.2 contains a classification of the business objects and the applicable management discipline which typically deals with the business objects listed.

Table 21.3 displays an overview of the goals of facility management on the strategic (long term), functional (technical or developmental) and day to day management levels.

21.5 Management aspects and business artefacts

Table 21.4 indicates typical business management activities relating to business artefacts.

21.6 Aspects of facilities management activities and processes

21.6.1 Operations for office spaces in buildings

The operational activities which form part of the general management of buildings include space management, budgeting and financial control, asset management as well as servicing management discussed below.

Management of spaces

To manage spaces an inventory of spaces with functional characteristics is required. The development of a unit cost reference data base relating to spaces and their use in a building is needed to support this activity.

Contracting and renting

The procedures involved in the contracting process between landlords and tenants typically require legal input.

Budgeting and financial control

Building owners and landlords evaluate typical cost elements such as building capital cost, municipal services, energy, climate control and air conditioning, property taxes, cleaning services, maintenance (short and long term), security and access control and insurance and manage these costs by allocating costs to activities, drawing up budgets with guideline deviations to be monitored and call on reporting on values and tendencies. The processes involved in invoicing rents and tenant debt management can be classified under this heading.

Asset management

Asset management deals with asset registers used for valuing, risk management and insurance and depreciation purposes and the general control of assets. Scheduling of maintenance and service requires reference to the asset register.

Table 21.1: Facility Management Functions

Function	Related disciplines/ Activities
Strategic and tactical planning	Property management and economics
Financial forecasting and budgeting	Financial, auditing, quantity surveying
Real estate procurement, leasing and disposal	Real estate, legal, property valuation
Procurement of furnishings, equipment and outside facility services	Interior decorating, facility planning
Construction, renovation and relocation	Architecture, Engineering and Economics
Health, safety and security and code compliance	Health, legal
Environmental issues	Environmental Engineering
Development of corporate facility policies and procedures	Strategic management
Building operations and maintenance	Engineering and facilities management, software engineering
Quality management, benchmarking and best practices	Management with engineering input
Architecture and engineering planning and design	Architecture and engineering
Space planning and management	Facility management
Management of support services e.g. transportation and catering	Facility management
Telecommunication	Engineering, utility management

Table 21.2: Relation between business objects and management disciplines

Object Class	Applicable management discipline
Physical infrastructure, energy and water supply, waste removal (Matter and Energy)	Facilities management
Portable business equipment (Matter)	Facilities management
Consumable stock supply (Matter)	Logistics and procurement management Refer to chapter 23
Procedure manuals, books, documents, computer software and intellectual property.(Information)	Document management Refer to section 21.8

Table 21.3: Engineers and Facility- and Practice Infrastructure Management

Level	Strategic	Functional	Management
Facet			
Goals	Optimal deployment of capital with an optimal return	Optimal development and upgrading of facilities	Optimal cost-efficient operation of facilities
Activities	Planning Tasks Selection of possible industrial, residential or business projects; Procure finance and know-how in general	Engineering Tasks Setting of design standards; Planning and detail design; Tenders and construction; System development and deployment	Management Tasks Operations; Maintenance; Risk Management; Financial Management; Information processing and reporting
Outcomes	Long term plan: Finance and development	Successful development projects	Management plan and procedures; Operational budget

Servicing and operational management

Aspects of the operation of buildings which need to be managed are classified under this heading i.e. security and access control, personnel facilities such as cafeterias and toilets, cleaning and solid waste disposal, energy and water supply, waste water removal, telecommunication, lighting, climate control and maintenance of indoor plants and gardens.

21.6.2 Monitoring and managing building subsystems

Some building subsystems such as climate control and energy supply need real time control activities with suitable reporting and feedback of data.

21.6.3 Maintenance

Planned maintenance as well as response maintenance are activities typically dictated by features designed and built into buildings.

Decisions on resources to be deployed for maintenance i.e. use of in-house personnel with suitable tools and equipment and support from outside service contractors need to be based on accurate quality of service and costing data.

Schedules for routine maintenance, planned maintenance and refurbishment need to be available from the onset of the service life of a facility to ensure that the expenses required are planned for.

21.6.4 Risks and exceptional events

Unforeseen events can have an impact the day to day activities in a building as well as the profitability of its operation. Typical events which can occur and will need immediate management attention include malfunction of system components within the expected service life, vandalism and poor quality maintenance. To deal with damage to facilities due to fire, natural disasters such as floods, earthquakes as well as political uprising suitable crisis planning is required.

The product of thinking under this heading will be a general risk management plan which includes an insurance portfolio plan and crisis management procedures.

Chapter 25 deals with risk management in general.

21.6.5 Health and safety

The facility manager needs to be conversant with the legal requirements and must ensure that buildings which the public have access to as well as buildings occupied by personnel comply with the applicable legislation and regulations.

21.6.6 Feedback from the operational environment to the planning and design environment

The importance of structured feedback from the operational environment to the planning and design environment to establish best practice principles regarding typical layouts, finishes and equipment configurations which perform satisfactory should be encouraged.

21.7 Software Implementation

Computerised information systems play an important supporting role in facilities management practice. The collection of reference and performance data as well as the processing and presentation of information for operating and management decision making is required.

Typical uses of information systems in facilities management are:

1. Maintenance of 'as built' and 'as is' drawings and data sheets
2. Management of space utilisation in facilities
3. Indexing and cross referencing of the facility component inventory for asset management and valuation purposes
4. Maintenance management i.e. planning, scheduling controlling and costing

5. Computerised financial systems which include costing, accounting, modelling and reporting functions.

The software technology which can be directly applied or adapted for use in facility management applications are listed below.

- General commercial systems i.e. spreadsheet, word processing and database systems.
- CAD - Computer Aided Drafting
- GIS - Geographic Information Systems
- MMS - Maintenance Management Systems
- Purpose made facility management software

21.7.1 Asset register

The asset register which typically forms part of a bookkeeping system provides the logical link from the facilities management to the bookkeeping system. Commercial software asset register implementations carry data on the capital value, depreciation, maintenance expenditure and write-offs which are linked to the general ledger for financial reporting.

21.8 Document management

Models for business document management consist of business objects and activities and processes for document management for a professional services business.

Burgers [20] defines document management systems as computerised software systems that will manage the creation, retrieval and indexing process of documents to secure and centralised repositories.

The articles by Burgers [20] and Albertyn and Fourie [2] contain further material on this topic.

21.8.1 Business objects for document management

The document objects which are dealt with in this section refer to completed internal documents developed for projects as well as documents brought into the business from external sources.

Documents can be broadly classified as relating to a specific project or to the business aspect of the organisation.

Table 21.5 lists the document media formats which are in use. Management decision making as to the format to be used for long term storage of information is required. Conversion of all material which needs to be archived to a standard format which will be maintainable and readable in the future needs to be considered.

References such as National Institute of Standards and Technology, [88], contain detail information on media formats and the applicable advantages and disadvantages of their use.

21.8.2 Document management business processes

Typical document management related activities and business processes for the professional services business include:

- Document procurement
- Document storage and security management
- Document indexing
- Library service support for obtaining documents located internally and externally to the business
- Document media format conversion
- Document archival at secure off-site storage facilities
- Document destruction

Table 21.4: Management activities relating to business artefact classification

<i>Artefact Class</i>	<i>Investigation, environmental scanning and forecasting</i>	<i>Planning and budgeting</i>	<i>Organising</i>	<i>Leading</i>	<i>Co-ordinating</i>	<i>Controlling</i>
Physical infrastructure, energy and water supply, waste removal (Matter and Energy))	Estimating requirements	Office planning and budgeting	Office facility organising	Leading office administration personnel	Interaction with landlords, local government and utility companies	Quality control of physical office environment
Portable business equipment (Matter)	Estimating requirements, costs and technology scan	Equipment planning and budgeting	Business equipment roll out and operation	Leading equipment operations and maintenance personnel	Business equipment operations and maintenance	Checking and machine performance monitoring
Consumable stock supply (Matter)	Estimating requirements and costs and product scan	Consumable stock planning and budgeting	Stock ordering, storage, requisitioning and distribution	Leading stock administration personnel and activities	Stock replenishment and ordering	Stock control costing and accounting
Procedure manuals, books, documents, computer software and intellectual property. (Information)	Development requirement scan, product scan, cost estimates	Development and supply planning and budgeting	Organising development and supply, storage, requisitioning and distribution	Leading development and administration activities	Coordinating distribution, storage and access control	Checking distribution, storage and access control

Table 21.5: Document media formats

Document media format	Encoding	Document type example
Text and images Printed on paper	Visible text and graphics	Letters, Articles, Drawings, Magazines, Books in bound paper format
Images printed on photographic paper	Visible images	Printed photographs
Magnetic data disks	Digital electronic	Computer data
Magnetic audio disks	Digital electronic	Audio (voice and music) on disk
Magnetic video disks	Digital electronic	Video on disk
Compact Laser Disks	Digital electronic	CD's and DVD's for data, audio and video material
Flash semiconductor memory devices	Digital electronic	Computer flash disks
Video tapes	Analogue electronic	Video cassettes
Audio tapes	Analogue electronic	Audio (voice and music)cassettes
Photographic microfilm	Micro images on film	Microfilm images
Photographic film and slides	Positive or negative images on film	Film negatives and 35 mm slides and cinema film formats

- Document copyright protection

The extent to which document collections belonging to personnel of a business need to be integrated into the document management system of the business is an issue which needs to be negotiated. Personnel can be supported to index personal satellite libraries which can then be linked to the central system and uncoupled again from the system as required.

21.9 Conclusions and recommendations

The review of the facilities and document management function of the professional services enterprise in this chapter indicates that facilities and document management systems for the enterprise can be modelled and structured using basic concepts discussed in chapters 4 and 7.

The implementation of expensive commercial facilities and document management software systems is seen as an ‘overkill’ for the typical professional service provider. Facilities management does not constitute the main goal of the business.

A facilities management outsourcing service as a parallel product offered by an engineering consulting group of course requires the deployment of software to support the service offered. An example of such a business is described at WSP Group [128].

Basic database driven facility and document systems can be developed to provide the required business process and management reporting functionality.

Chapter 22

Knowledge and Information Management

22.1 Introduction to knowledge management

Knowledge Management (KM) refers to a range of practices used by organisations to identify, create, represent, and distribute knowledge for reuse, awareness and learning across the organisation. Knowledge Management programs attempt to manage the process of creation or identification, accumulation, and application of knowledge or intellectual capital across an organisation. Wikipedia [126].

Knowledge management focuses on the management of artefacts which are data and information related.

In a professional service business a large proportion of the knowledge is seated in the highly trained technical personnel employed by the organisation. It is a challenge for the management of such organisations to implement strategies to extract, distill and store parts of the intellectual capital outside selected individuals. This will reduce the risk of concentration of knowledge and the effect on the knowledge available when personnel leave the organisation.

22.2 Models for business knowledge management

Models for business knowledge management consist of business objects and activities and processes for knowledge management for a professional services business environment are discussed here.

22.2.1 Business objects for knowledge management

A listing of selected business objects for knowledge management is given in table 22.1. A classification of technical disciplines with which a business interacts needs to be drawn up and standard keywords defined which can be used on project records and library catalogues to support retrieval of required information.

Table 22.1: Business knowledge management objects

Knowledge management object	Linked management activity or processes
Intellectual capital	Identification and storage
Structured technical discipline database with keywords	Project registration
Library classification e.g. Dewey system	Book and media sourcing and purchase
Project information source reference	Project diary or web site
Practice manual / Project manual	All project production related work and processes
Business procedure manual	All business functions and processes
Candidate technique listing	Environmental scanning for technology and business techniques
Trademarks, registered designs and trade secrets	Trademark and design identification and registration as required
Copyright and patents	Patent identification and registration as required

The Practice or Project Manual

A professional service business should adopt the approach of a learning organisation and make a long term investment in the development of a practice or project manual.

The practice manual should also contain standard procedures as well as formats of forms and other documents for project, contract and construction administration.

The business procedure manual

A business procedure manual can be developed in parallel with a practice manual.

It should contain all standard procedures applicable to the business. The manual can be organised by business function or any other logical structure relating to the business. Standard formats of business forms and documents should be contained in the document. The manual contents can be made available on a business intranet for easy access.

The manual should be an invaluable resource to support the induction and training of new personnel.

22.2.1.1 Software systems

Software systems and the datasets associated with them can be useful sources of project and business information. To make this source available ‘Search engine’ type software which can search company datasets can be useful.

22.3 Knowledge and information for project execution

Wikipedia [126] states that individuals undertaking a new project for an organisation might access information resources to learn best practices and lessons learned for similar projects undertaken previously, access relevant information again during the project implementation to seek advice on issues encountered, and access relevant information afterwards for advice on after-project actions and review activities.

Knowledge may be captured and recorded before the project implementation, for example as the project team learns information and lessons during the initial project analysis. Similarly, lessons learned during the project operation may be recorded, and after-action reviews may lead to further insights and lessons being recorded for future access.

The key to accessing this information is a well structured keyword driven system linked to project records. Formal use of project diaries which can be placed on project web sites will ensure that relevant information is recorded. Knowledge management systems, repositories, and corporate processes to encourage and formalise these activities are required.

22.4 Protecting business artefacts against misuse

Protecting business artefacts which include knowledge can be achieved using trade marks, registered designs, trade secrets, copyrights and patents as required.

Business logo’s and trade marks can be registered to ensure that unauthorised use by other parties does not occur. Unique designs can also be registered if required.

A business should guard against copyright infringements. Using copyrighted material can lead to legal actions. On the contrary action against parties misusing copyrighted business material can be considered. In-house developed software products are also subject to copyright.

When innovative designs or products are developed patenting is a way of ensuring that other parties do not provide similar products for a set period of time.

Employment and share holder contracts should contain clauses which state the way a person linked to the business is allowed access to copyrighted or other protected material while in employ of the business. It should also cater for procedures to be followed when the person leaves the company.

22.5 Conclusions and recommendations

The review of the knowledge management function of the professional services enterprise in this chapter indicates that knowledge management systems for the enterprise can be modelled and structured using basic concepts discussed in chapters 4 and 7.

A high level business goal to develop, document and protect business production (project) knowledge as well as business procedure knowledge is required.

Basic database driven knowledge systems can be developed to provide the required business process and management reporting functionality.

Chapter 23

Logistics

23.1 Introduction to logistics for the professional service business enterprise

Mazda [81] states that logistics is the total concept encompassing the flow of goods from the supplier through the manufacturing plant to the customer. It covers aspects in the manufacturing environment such as procurement, goods receiving, work in progress, stock control, finished goods stores and distribution to the customer. Techniques such as just in time (JIT) and materials requirement planning (MRP) are applicable in this environment. An indirect link from logistics to quality management e.g. total quality management (TQM) techniques exists.

A dictionary definition of logistics reads: ‘The time related positioning of resources’ (Wikipedia [126]).

The professional service business is an information processing and document manufacturing business and production logistical considerations revolve around this.

In parallel to the document manufacturing taking place, office supplies, consumables and stationery needs to be sourced and made available to personnel when required.

23.2 Models for logistics management

Models for logistics management consist of business objects and activities and processes for logistics management for a professional services business.

23.3 Business objects for logistics management

Certain business objects require logistics management processes.

Business objects which could benefit from a logistics approach to management are listed in table 23.1.

23.4 Professional Service Business Logistics Activities and Process

A professional service business deals with inbound logistics as well as with outbound logistics for selected business objects. The terms inbound logistics and outbound logistics are used as primary business value adding activities in the Porter value chain model shown in Glad and Becker [42, Figure 2.1].

Table 23.1: Business objects requiring logistics management

Business object	Logistics management process required
Office stationery	Typical material supply and stock holding management
Copier, printer and facsimile machine consumables	Typical material supply and stock holding management
Project documents	Requirements scheduling with internal project planning Input information and material Internal content quality control

Table 23.2: Project document classification

Project document type	Description
Letter report	Short report on technical query
Feasibility study	Project evaluation before detail planning
Project planning report	Report defining design parameters and design execution
Project design report	
Bill of quantities	List of items to be priced for budgeting or tender purposes
Project specification	Document describing project component requirements
Contract tender document	Document containing conditions of contract, project specifications, bill of quantities and tender drawings
Tender evaluation report	Detailed evaluation of tenders received with recommendation to client on appointment of construction agent
Construction progress report	Monthly report to client on project progress and cash flow
Project completion report	Report summarising actions and events through project life cycle

23.4.1 Inbound logistics

Inbound logistics management processes for project document production as well as for office supplies and consumables are required.

General activities relating to inbound logistics are:

- Checking of goods received to ensure complying with order quality, quantity and type.
- Returning defective goods for replacement or credit.
- Storage of stock in storage spaces where deterioration, safety and risk issues such as theft and fire need to be dealt with. Refer to section 25.4.1.
- Distribution of material to satellite offices or personnel in remote locations.

Logistics management links to the financial systems via the order processing logic. Refer to section 18.8.

Project Document Production Logistics

The purpose of production logistics is to ensure that each personnel member and workstation has access to and can receive suitable, quality and correct information at the right point in time.

Production logistics techniques can be applied to project document processing. Production logistics provides a means to achieve labour input efficiency in the production of project documents.

Typical types of project documents are listed in table 23.2.

Project document requirements e.g. special covers, paper, binding, printing, media and requirements for electronic digital storage space can be deduced from the type and size of the documents required and planned for a project.

Timing requirements to be input into the logistical process can be derived from the project activity schedule.

The process of compiling a project document can be done in an optimal way and should be streamlined if inputs into the compilation process are managed well.

Office consumables and stationery

The costing of office consumables and record keeping on stock levels and item and product usage can form the basis of management reports. Identification of trends which cannot be readily explained should lead to management intervention and action to reduce office overhead costs.

The extent to which consumables used can be recovered as project costs needs to be investigated to reduce administrative overheads.

23.4.2 Outbound logistics

Goods and services which are outbound in the professional service environment include basic communication objects such as e-mail messages, letters, letter reports, project reports, contract documents and other. Refer to table 23.2 for a classification of project documents.

Outbound logistics relating to project document manufacture is required. To ensure quality control of such documents the development of standard formats should be considered and an office registry system linked to the document management system is required to track and trace the flow of these documents. Reliability and timing of delivery is important and ensured.

Specialised e-mail logging and reporting software solutions are available on the market to support the document register requirement in the digital electronic communication environment.

The development and maintenance of project web sites as part of the brief for a project can be useful in planning, controlling and tracking the flow of documentation for a project.

23.5 Client Project Logistics

Consulting engineering professional service procuring clients can require the project consultant to provide a service extending beyond the traditional project planning, design, specification documentation, tendering and construction monitoring scope.

The successful construction and commissioning of large scale industrial and mining projects require tight integration of the planning, design construction and plant and sub-assembly procurement processes. To achieve this, the procurement aspect of a project can also be outsourced to a professional service provider such as a consulting engineer.

In the scenario outlined above the consultant needs to become involved in logistical planning, scheduling, procurement and installation linked to the construction process.

The deployment of specialised knowledge and logistical techniques in this case falls outside the scope of the study documented here.

23.5.1 Supplier management

Supplier management in the logistical context refers to the process of shopping for suitable quantities, quality at an acceptable price. Special discounts can be negotiated with preferred suppliers.

23.5.2 Materials control

The level of business finance locked into the material required for office stationery and consumables as well as project document production requirements should not warrant specialised stock control measures. If large volumes of material needs to be handled this can however be considered.

23.6 Conclusions and recommendations

The review of the logistics management function of the professional services enterprise in this chapter indicates that logistics management systems for the enterprise can be modelled and structured using basic concepts discussed in chapters 4 and 7. Basic database driven systems can be developed to provide the required business logistics process and management reporting functionality.

Service businesses which provide specialised laboratory services need to pay special attention to the logistics function of that part of the business.

Chapter 24

Administration

24.1 Introduction to the administrative function

According to Du Plessis [32] and Du Plessis [33] the administrative management function in business involves the creation of systems, procedures and techniques by which information can be acquired, classified, processed, stored and then disseminated rapidly and timeously to the parties involved so that decision-making and planning are enhanced. It is specifically designed to provide services to other departments of a business enterprise.

The administrative function can be organised as a separate entity in a business or can be grouped with the applicable other business management functions as required. In smaller enterprises all administrative functions will typically be performed by one grouping of persons or one department.

The publication by Macleod [78] contains information on the administrative requirements of starting a business in South Africa.

24.2 Models for administrative management

Models for administrative management consist of business objects and activities and processes for logistics management for a professional services business.

24.3 Business Objects and Business Administration

Business objects which are involved in the administrative process include a wide variety of documents.

As an example a list of selected business documents with the period for which they legally are required to be retained in a business is listed in table 24.1. A complete list can be found in Van der Merwe et al. [122, Appendix H].

A set of prescribed recurring actions and procedures are also required from the administrative function. If the process complexity warrants detailed control of these functions, they can be modelled using the engineering process model described in chapter 11. Other suitable process models described by Huhnt et al. [62] and Huhnt [59] can also be used.

Du Plessis [32] confirms that the effective and reliable processing, storage and dissemination of administrative data can be crucial in the making of important decisions that affect the future of a business. Administrative and reporting needs need to be identified accurately so that the administrative function can be organised accordingly.

Business administration requirements depend on the form of the business. Publications such as Van der Merwe et al. [122] describe the legal administrative requirements for corporate business administration. The publications are regularly updated to reflect changes in the legal and regulatory environment. Van der Merwe et al. [122, Appendix E] contains a table of duties imposed by the South African Companies Act on directors of companies and Van der Merwe et al. [122, Appendix F] lists penalties for contraventions of the Companies Act, 1973 as amended.

The basic legal forms that a business entity can take in South Africa are listed in table 24.2. All the business forms listed can be used for the professional service enterprise. Legal requirements set by clients as well as legislation such as the Engineering Professions Act 46 of 2000 [47] need to be taken into account when deciding on the legal form a business must take.

24.4 Interaction between the administrative processes and other business functions

The administrative requirement relating to each basic business management function is set out below per business management function.

24.4.1 Business Strategy and Policy

The administrative input to this function is that of record storage, recovery and dissemination as required. Business strategy documentation, practice manuals and related documentation needs to be stored for retrieval and modification as required.

24.4.2 Marketing Administration

Aspects of marketing administration are discussed in chapter 17.

Administrative functions related to marketing include:

- Maintenance of client and customer data bases which are typically linked to the project database as well as the debtor database.
- Records of marketing actions such as visits to clients and customers, attendance of symposia and meetings.
- Organising of special events.

The project system should be used to log all project proposals made or participated in and the project status can be changed from proposal to fee or time based project when a proposal progress to be a full scale project.

24.4.3 Financial Administration

Financial administration is an ongoing process linked to the regular monthly cycle of customer project invoicing, creditor payments, stock taking and debtor administration.

Some organisations elect to maintain personnel debtor and creditor systems to control the flow of payments and disbursements between the business and the personnel for e.g. travel, accommodation and entertainment. Company credit cards issued to selected personnel can be involved in administering this exchange of payments.

Interaction with bookkeepers and auditors to supply information for and receive e.g. annual financial statements is required.

24.4.4 Personnel Administration

All aspects of personnel administration are dealt with in chapter 19. The basic requirements on information to be retained in this process is summarised in table 24.1.

24.4.5 Facilities Administration

Facilities administration includes document as well as library administration.

Facility documentation, e.g. purchase and leasing documents as well as warranty and guarantee information of equipment in use needs to be kept up to date. Scheduling of maintenance can be done as part of the administrative aspects of facility management.

Service business enterprises need to maintain a technical library which can include book type documents as well as digitally stored documents which need to be referenced during the normal day to day production and other activities of the personnel. The administrative procedures in and around a library needs to be attended to.

Document administration linked to the document management system will ensure that project and other correspondence, reports, tender and contract documents as well as drawings and media items such as photographs and videos are stored and archived as required by project agreements or other legal requirements.

Modern information system technology supports the development of a business intranet. A typical intranet system provides logically ordered access to business documents using a web-browser interface at

the personal computer workstation. Personnel can then access documents which they are permitted to use as required.

The information technology infrastructure administration function also needs to be attended to.

Off site storage of key documents as well as backup copies of critical digital datasets needs to be arranged and administered.

Fax machines, office copying machines, digital printers, plan printers and other machines need to be supplied with consumables as required and records of production and service performance need to be kept.

24.4.6 Logistics Administration Function

The stock taking administrative activity links logistics administration with financial administration. Financial administration requirements are summarised in section 24.4.3.

24.4.7 Project Administration Function

The project administration function needs to ensure that the following activities in a around projects are completed:

- Project registration and take on data completion and approval
- Project take on feedback to client
- Project budgeting completion and approval
- Project activity planning and resource scheduling
- Project invoice scheduling and generation as required
- Regular project progress reporting and dissemination of report sheets
- Project procurement process i.e. tender advertisement, tender document production and distribution, tender closure administration and contract award administration as required by the client.
- Project progress certificate processing and approval of contractor and suppliers invoices for payment by the client
- Project closure reporting and document archiving

24.5 Systems support for administrative management

Database functionality outlined in chapters 7 and 13 provides adequate support for the administrative function. Some triggering / alarm mechanism can be supplied to trigger certain routine administrative tasks.

24.6 Administrative Function Reporting Requirements

An administrative management reporting model with suitable structures based on the concepts defined in chapter 12 can be defined. Reports drawing data from enterprise databases can be defined as required.

24.7 Conclusions and recommendations

The review of the administration management function of the professional services enterprise in this chapter indicates that the Engineering process model is suited to support the project administration requirement for the core production process of the business.

Basic database driven systems can be developed to provide the required administrative process and management reporting functionality not already available in other business systems or sub systems.

Table 24.1: Disposal and Retention of Business Documents

Document type	Required retention period
Close corporation founding statements and minutes	Indefinite
Close corporation annual accounting records	15 years
Company founding documents and minutes	Indefinite
Register of directors and members	15 years
Share registration documents	3 to 15 years
Company annual financial statements	15 years
Invoices and periodic accounting reports	4 years
Contracts and agreements	5 years after expiry
Banking records	4 to 6 years
Employee records	3 to 4 years
Unsuccessful applications for employment	1 year
Payrolls, salary registers, tax returns	3 to 5 years
Donations granted	4 years
Insurance accident reports and claims correspondence	3 years after settlement
Pension records - actuarial valuation reports	10 years
Property records such as agreements and leases	4 to 5 years
Property title deeds	Indefinite
Shipping documents	2 years after shipment completion
Taxation VAT records, tax returns and assessments	5 years
Technical and research records	Depends on conditions
Microfilm records	Indefinite
Electronic archival and imaging systems records	15 years
Patent and trade mark records	Indefinite

Table 24.2: South African Business Legal Forms

Business legal form	Notes
Sole proprietorship	Business owner conducts business in his personal capacity
Partnership	Format defined in partnership agreement
Close corporation	1 to 10 members
Incorporated company	Directors exposed to unlimited liability
Private Company	Up to fifty members
Public Company	Seven or more shareholders with company shares listed for trading on the stock exchange
Trading trust	Format defined in trust document and controlled by trust legislation

Chapter 25

Risk Management

25.1 Introduction to risk management for the professional service business enterprise

Risk management deals with the planning, organising and controlling of events, activities and resources in order to minimise the impact of uncertain events.

In this chapter aspects of risk management relating to professional services businesses are discussed. Risk management in this environment focuses on two aspects i.e.

- The risks of project execution which are projected onto the service business
- The business enterprise risks

Risk management for financial risks are discussed as well.

Further material on risk management for enterprises and projects can be found in i.e. references such as Burke [21], Nicholas [87], Scarborough and Zimmerer [107] and Bester and Koch [16].

25.2 Models for risk management

Models for risk management consist of business objects and activities and processes for risk management for a professional services business.

25.3 Business objects and risk management

The business objects involved in risk management include the project being executed by the business as well as the business functions/processes and infrastructure required to execute the projects.

25.4 Risk management activities and processes

The process of risk management can be divided into:

- Risk identification and evaluation
- Risk analysis and quantification
- Risk allocation and control

25.4.1 Risk identification

Risk identification focuses on business function risk identification and business infrastructure risk identification.

Business function risk identification

Business function risk identification can be done function by function and structured reports for periodic review can be drawn up.

Typical business functions, with typical risks which can be identified for a function, are listed below.

1. Marketing function risk identification. Apart from the effect of incorrect marketing intelligence or other erroneous information input into the marketing management process the marketing management approach discussed in chapter 17 is designed to deal with the inherent uncertainties in the marketing of professional services.
2. Finance, bookkeeping and auditing risk identification. Financial risks are discussed in section 25.5.
3. Personnel function risks. Typical risks which can be identified here include over and under staffing, dependence on key persons, possible fraudulent activities of personnel, misrepresentation of qualifications and experience and in service injury, long term illness and incapacity to perform duties.
4. Facilities management risk identification. Risk management relating to business facilities is discussed in sections 25.4.1 and 25.4.4.
5. Knowledge management risk identification. Risks relating to knowledge management which can be identified include unavailability of knowledge and technology to deal with projects at hand, outdated knowledge in place and exposure to knowledge taken over by competitors due to problems with registering of copyright, intellectual property and patents.
6. Logistics risk identification. Logistic risks in the professional service business should not represent a major problem. Material required to perform tasks should be readily available for purchase or rent. Where offices are located in remote areas and personnel need to work on inaccessible sites special attention may be required here.
7. Production risk identification. Production risks are project risks which are discussed in section 25.6.
8. Administration risk identification. Administration risks originate from legal and regulatory requirements applied to business entities.

Business infrastructure and stock risk identification and evaluation

Structured reports linked to asset and infrastructure management systems as well as document management systems can be used for risk identification.

A high level classification of the business risks refer to:

1. Office accommodation and office infrastructure.
2. Goods in storage for use in the production and administration processes.
3. Company owned vehicles for transport.
4. Specialised laboratories and on-site testing equipment

An evaluation of business infrastructure risks such as those listed below are required. The impact of events on the day to day activities of the business and a possible interruption of such is also required.

1. Loss and theft
2. Accidents
3. Fire
4. Liability to third parties

25.4.2 Risk analysis and quantification, Risk allocation and control

Risk quantification, allocation and control focuses on business function risk quantification, allocation and control and business infrastructure risk quantification, allocation and control.

25.4.3 Business function risk analysis, quantification, allocation and control

1. Marketing risk quantification and control. The quantification and control of marketing risks are discussed in chapter 17.
2. Finance, bookkeeping and auditing risk quantification and control. Financial risks are discussed in section 25.5
3. Personnel function risks quantification and control. These risks can be listed in formal reports drawn from the personnel register and relying on supervisors to report any perceived or actual problems arising. Key person reliance can be evaluated and key person insurance bought as required. Statutory required insurance such as unemployment insurance (UIF) and Workman's Compensation Act (WCA) levies apply here as well.
4. Facilities management risk quantification and control. Risk management relating to business facilities is discussed in sections 25.4.1 and 25.4.4.
5. Knowledge management risk quantification and control. Risks relating to knowledge management can be quantified by studying the scope of envisaged projects. The unavailability of knowledge and technology to deal with projects at hand can be solved by relying on specialist sub consultants and service providers. The protection of knowledge from access and use by competitors can be dealt with via registering of copyright, intellectual property and patents as required.
6. Logistics risk quantification and control. The logistics risks identified should be dealt with by ensuring proper operations of supply processes as well as stock level control for basic office requisites and stationery.
7. Production risk quantification and control. Production risks are discussed in section 25.6.
8. Administration risk quantification and control. A formal review of applicable documentation with the aid of specialist consultants such as the company auditors should identify the requirements and risks relating to non-adherence to these requirements. Where business entities are situated in different countries special attention needs to be paid to these requirements.

25.4.4 Business infrastructure risk quantification, allocation and control

Business infrastructure risks can be dealt with in a number of ways. Risks can be avoided, reduced or controlled in a number of ways.

25.4.5 Risk control

Once risks have been identified they can be systematically reviewed and measures to control them be decided on. Typical measures include drawing up of and reviewing and checking of personnel training, equipment operation manuals, office procedure manuals. Once these options have been evaluated a plan of action can be drawn up, approved and implemented.

25.4.6 Risk avoidance and risk reduction

Risk reduction implies reducing the level of risk associated with an activity of reducing the number of activities with associated risks to be executed in house.

Legal documentation allocating responsibility for activities can also be reviewed to make sure that risks are properly allocated.

Risk reduction can be achieved by allocating risky activities to other better qualified and equipped service providers outside an organisation. E.g. quality control of ground fill compaction using radio active isotope based methods can be sub-contracted to service providers.

25.4.7 Risk financing, retention, transfer and insurance

Risk control and loss reduction measures imply risk financing through risk retention or transfer and insurance.

Risks can be transferred by transferring the activity which contains the risk or by transferring the financial loss of a risk occurrence via e.g. contract conditions. The party contracting to provide the activity will then price the risk element into its price.

Risk costs can be financed by charging losses to operating costs and expenses, making up front provision for losses through contingency funds or insurance arranging loans to spread the cost of the loss over a period of time. The evaluation of the typical quantum and possibility of recurrence of a loss associated with a risk will point to the best approach of risk financing to be adopted.

Insurance provides a mechanism for handling risks with a low probability of suffering a large financial loss which an organisation cannot afford to retain itself.

25.5 Financial risk management processes

This section deals with the identification of service business enterprise financial risks as well as the analysis, quantification, allocation and control of these risks.

25.5.1 Identification of financial risks

A well designed auditing process should identify and quantify the financial risks to which a business is exposed. The major risks in the financial category which need to be identified are listed below.

1. Solvability risks. The total, long term as well as working capital asset / liability balance for the business needs to be actively managed to ensure that solvability ratios (ratios of total as well as selected classes of assets and liabilities) are met. These ratios are usually defined according to the policy set by the owners, partners, shareholders or board of directors of the business.
2. Capital structure risks. Capital structure risks are typically inherent in the service type business. These businesses are funded by loans supplied by share holders. The capital is typically limited and only matches the creditor and personnel payment requirements for a period of a few months. It could be that high value assets such as property are held in the business.
3. Cash flow risks. The business needs to be in a position to make payments as required.
4. Short term loan and bank overdraft risks. Disciplined cash flow management should alleviate these risks.
5. Work in process and invoicing risks. Invoicing is the process of drawing up a document to serve as a claim for payment to be made by a client. In the professional service business this requires the balancing of project work in process entries (project costs) with the pricing formula for the project set out per invoice. Write ups and write offs can be required in this process.
6. Personnel productive time allocation risks. The allocation of personnel time to activities such as management, marketing and administration versus productive project time needs to be identified.

25.5.2 Financial risk analysis, quantification, allocation and control

To analyse and control financial risk in general financial reports and documents need to be monitored on a regular basis and management intervention needs to take place where targets are not met.

Analysis and control of some risks which need special attention.

1. Solvability risk analysis and control. Asset / liability ratios are usually defined according to policy documents need to be monitored on a regular basis and management intervention needs to take place where targets are not met.
2. Capital structure risk analysis and control. Close attention to cash flow management should alleviate the pressure on the capital structure of the business. If the cash flow experience changes over time adjustments will have to be made in the capital structure of the business. High value assets held in the business such as property should preferably be held in separate business entities to prevent problems of financing shareholder loan redemption when shareholders resign or retire.

3. Cash flow risk analysis and control. Cash outflow to creditors, personnel remuneration and shareholder payments need to be balanced with cash inflow from debtor payments, loans and other sources of cash income. The cash flow budget is the key document required to control this process. Formal payment timing agreements with clients should be in place. Debtor statements should be produced on a regular basis and non-payment followed up as required.
4. Short term loan and bank overdraft risk analysis and control. The risk of defaulting on loan interest and instalment payments as well as bank overdraft requirements is closely linked to the cash flow of the business.
5. Work in process and invoicing risk analysis and control. Disciplined control of the invoicing process will ensure that work in process entries are timeously converted to invoice items. The key document which supports this process is the project invoicing budget. This is linked to the cash flow budget. The process of write ups and write downs/off of work in process items needs to be carefully controlled by project managers as well as departmental and office managers assigned to this task. Work in process items such as personnel time booked on a project and project disbursements are volatile and are typically lost when written off. Transfer of time booked to a project, which cannot be invoiced to another project, is typically not possible.
6. Personnel productive time allocation risk analysis and control. The allocation of personnel time to activities such as management, marketing and administration should be managed carefully to ensure that personnel resources which are required for project work are available.

25.6 Project Risk Management Processes

Projects form the production units of a professional service business enterprise and are as such a major source of risk for the enterprise. A project failure can have serious and even detrimental consequences for the professional service enterprise. Project risks need to be identified with the client and construction contracting parties and steps taken to analyse these risks.

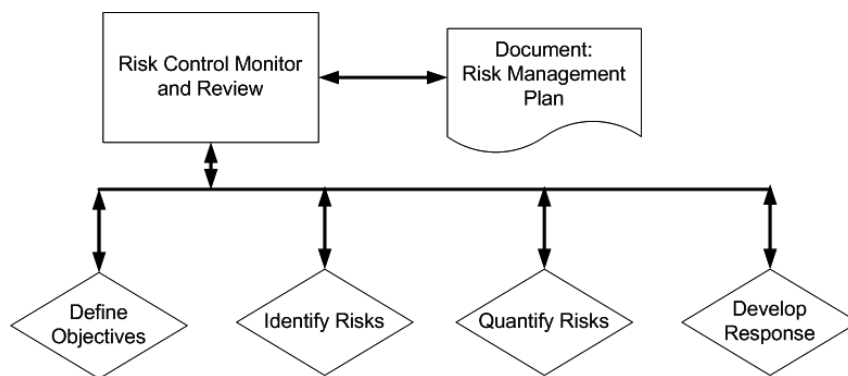


Figure 25.1: *Project risk management model*

Figure 25.1 shows a project risk management model given by Burke [21].

Once again the activities defined are:

- Define objectives: the context of the project and success requirements are defined.
- Identify risk: the areas of risk and uncertainty are identified which may limit or prevent the achievement of objectives.
- Quantify risk: the levels of risk and uncertainty are evaluated and prioritised according to impact and possible frequency of occurrence.
- Develop response: the response to the risks are defined i.e. eliminate, mitigate, transfer or accept.
- Document: a *risk management plan* documents the proposed way in which the risk on a project will be managed.
- Risk control: This function implements the risk management plan.

25.6.1 Project risk identification

A well written internal project manual which leads a professional in the project design, specification and documentation as well as the construction activity monitoring and quality control phases is essential.

25.6.2 Project risk analysis and quantification

Project risk is typically dictated by the level of experience with the specific type of project of the professionals taking part in the project. This also applies to role players and personnel from the client, consultant, constructor and material and related service delivery organisations involved in the project.

Design and build as well as ‘fast track’ design while build approaches to project execution have a major impact on project risk profiles which need to be analysed and quantified in detail.

Public-private partnership projects as well as private concession based supply of public infrastructure impacts on the financial risk of projects which needs to be quantified.

The way in which risks identified are to be dealt with need to be documented formally to ensure that risks are allocated as planned.

Client-consultant and client-service provider contracts should be reviewed to ensure that risks are allocated as intended.

Risk management responsibility also needs to be allocated formally and documented as such.

25.6.3 Evolution of Risk Through Project Life Cycle

Figure 25.2 adapted from Burke [21] shows the evolution of project risk through the project life cycle.

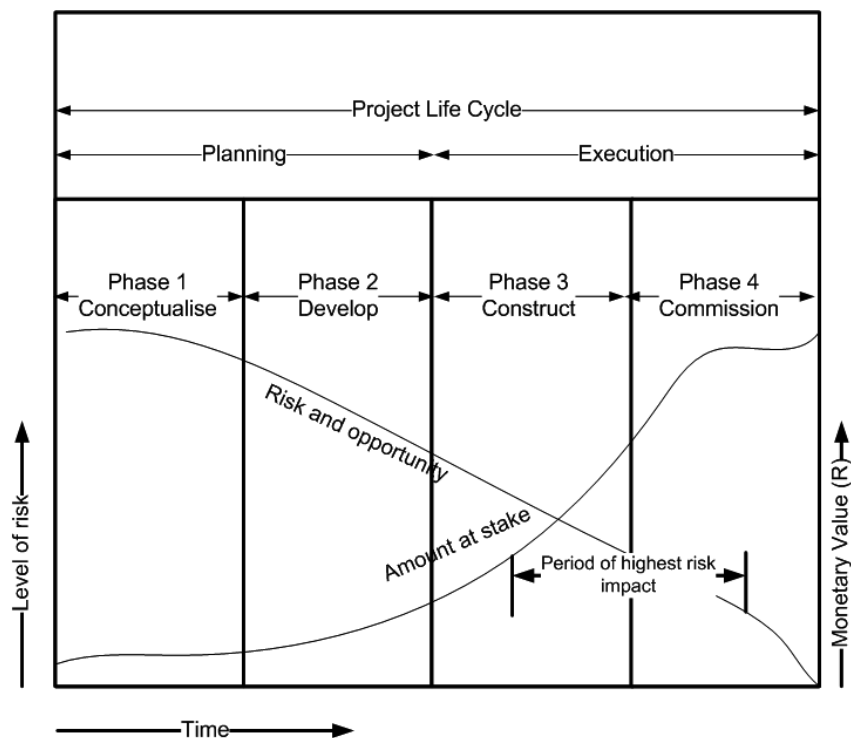


Figure 25.2: Project risk management model

Risk and opportunity are high at the inception of the project, and the greatest degree of uncertainty about the project outcome and future exists. As the project progresses the uncertainty decreases as decisions are made and project parameters are fixed and designs completed. This risk eventually reaches zero when the project is completed. The monetary amount at stake on the other hand starts out low and increases along the familiar S-curve as the project develops through its stages. The highest vulnerability to risk occurs in phases 3 and 4 as shown when the impact of adverse conditions will be the most.

The level of risk and financial exposure associated with a project when viewed from the perspective of the professional service provider can be deduced from figure 25.2 by determining the level of involvement and exposure to each phase of the project.

25.6.4 Project Risk Allocation and Insurance

The requirement by clients and especially public sector clients that professional firms need to provide adequate professional indemnity insurance to cover the project design in perpetuity and specification risk in total is partly being replaced by specific project related risk cover.

The pricing of professional indemnity insurance which has increased over the recent past due to claim experiences by insurers, has been the main reason for this trend.

25.7 Disaster Recovery Planning

Burke [21] defines a disaster as a sudden, unplanned catastrophe that prevents a company from providing its critical business functions for a period of time resulting in significant damages and losses. The time factor will determine if a problem of service interruption is an inconvenience or a disaster. Losing the electrical power supply for an hour may be inconvenient but to be without power for a month could lead to financial disaster.

The objective of disaster recovery management as a contingency response from the risk management plan is to reduce the consequences of a disaster to an acceptable level. A disaster recovery plan should be in place so that in the event of a disaster a team can control the plan and implement it quickly and effectively. Elements of the military approach to operations are needed here.

Elements of a disaster recovery plan are:

1. Disaster recovery team setup and muster procedure
2. Informing protection and rescue agencies, insurance brokers, clients, suppliers, media and other affected parties
3. Arrangements for office relocation if required
4. Arrangements for emergence sources of power, water and other services
5. Information system recovery

25.8 Risk management manual and standard report contents

A professional service business should invest in the development of a dedicated risk management manual or add a risk management section to its practice manual. This should then be reviewed with existing and new personnel to ensure that the way in which the business handles risks according to the defined policies and procedures are well understood.

A formal incident reporting procedure should be in place. The incident log should be reviewed and analysed on a regular basis.

Insurance claims made logged and documented for future reference.

The risk classification outlined in the sections above implicitly define a logical set of risk management reports which can be designed. These reports can be structured with links to the accounting, document management as well as facilities management systems as required.

The structures of the reports referring to management, office or departmental as well as project logical hierarchies can be done using graph processing techniques described in chapter 12.

25.9 Conclusions and recommendations

Risk management with suitable systems deployed is an important business function for the professional service business.

Chapter 26

Practice Management Systems

This chapter provides a short review of major management software systems available on the market.

A review of the monetary impact of software systems on businesses was given by Beucke [17]. Accordingly engineering enterprise expenditure on software products can be classified as follows:

Type of computer application or system - pricing units

Desktop computer operating systems - 1 000 (10^3)

Structural analysis technical software - 10 000 (10^4)

Computer aided drafting (CAD) systems 100 000 (10^5)

Business management systems - 10 000 000 (10^7)

It is clear that management software is currently extremely expensive, consequently great care is required in the acquisition and implementation of such software.

Some commercially available business management and professional practice management systems are listed in this chapter for reference purposes.

26.1 ProMan by Akron Software

ProMan developed by Akron Software, Greyling [50], was originally developed for Steffen, Robertson and Kirsten Inc. Consulting Engineers from 1985-1987. The product was commercialised and has 35-40% market share in South Africa and a presence in Hong Kong and Australia.

According to the Akron Software web site:

‘ProMan is a completely integrated project costing and financial management system designed for the optimised management of practices in the professional services industry. It was developed specifically to improve efficiency and profitability in multi-disciplinary professional service companies by allowing comprehensive business unit based accounting.

The system is currently in use in South Africa, Hong Kong, Malaysia, Thailand, Zimbabwe and Botswana. In South Africa ProMan was particularly successful. The latest MIS survey of the South African Association of Consulting Engineers (SAACE) shows that 38.9% of the Consulting Engineers registered with SAACE use ProMan. In Australia ProMan is marketed under the trade name ‘PinPoint’.

26.2 Systems by Deltek Inc.

The Deltek, Deltek Systems Inc. [29], practise management product was implemented by Stewart Scott South Africa in 2003 in a migration process from the ProMan system.

According to the Deltek web site:

‘Deltek provides software solutions specifically designed to meet the needs of project-driven businesses. Today our software applications and solutions help more than 11,000 organisations achieve success worldwide.’

26.3 SAP

SAP is a well known Enterprise Resource Planning (ERP) system used in the manufacturing and local government sector in South Africa. The co many operates on a worldwide basis.

According to the SAP web site:

‘Founded in 1972 as Systems Applications and Products in Data Processing, SAP is the recognised leader in providing collaborative business solutions for all types of industries and for every major market.’

‘mySAP Business Suite applications are based on the SAP NetWeaver platform, an integration and application platform. This reduces total cost of ownership across the entire IT landscape and supports the evolution of mySAP Business Suite to a services-based architecture.’

The mySAP suite consists of:

- mySAP Customer Relationship Management
- mySAP ERP
- mySAP Product Lifecycle Management
- mySAP Supply Chain Management
- mySAP Supplier Relationship Management

Refer to SAP Aktiengesellschaft [106] for more detail information on SAP and related systems.

26.4 Dynamics / Business Solutions / Great Plains by Microsoft

Microsoft offers a suite of business and management software solutions now named Dynamics. Previous versions were named Business Solutions and Great Plains.

The system components are built on the *Microsoft Windows Server* and *SQL Server* platforms. According to the Microsoft Dynamics web site:

‘Microsoft Dynamics solutions include a number of product families. The applications within these product families address the following business needs:

- Business Intelligence and Reporting - Manage budgets, create and consolidate reports, forecast more accurately, and look for trends and relationships in any part of your business.
- Collaborative Workspaces - Strengthen employee productivity, as well as relationships with partners and customers, by providing secure, Web-based access to appropriate data through portals and e-commerce functionality.
- Customer Relationship Management - Manage customer groups, create and launch marketing campaigns, track customer activity, and organise sales and after-sales. Help field staff serve customers more efficiently.
- Financial Management - Control general ledger, payables, receivables, inventory, sales process, purchasing, fixed assets, and cash flow. Perform reconciliation and collections.
- HR Management - Manage human resources from mapping, recruitment, and employee registration, to skills development and processing of payroll and benefits.
- Manufacturing - Coordinate your entire manufacturing process from product configuration and supply and capacity requirements planning, to scheduling and shop floor.
- Project Management - Manage resources, forecast costs and budgets, track time and expenses, and organise contracts and billing.
- Retail Point of Sale - Run retail operations from point-of-sale to delivery. Increase customer flow, speed up lines and tasks, control inventory, and automate purchasing.
- Supply Chain Management - Organise single or multiple site warehouses; handle order promising, demand planning, and online collaboration with suppliers. Track distribution; inventory, order, and purchasing management; sales forecasting; and warehouse management.’

Refer to Microsoft Corporation [83] for more information.

26.5 PeopleSoft and JD Edwards by Oracle

PeopleSoft and JD Edwards Products are now marketed and supported by Oracle Corporation
The PeopleSoft and JD Edwards product lines are listed below.

- PeopleSoft Enterprise
- JD Edwards EnterpriseOne
- JD Edwards World

Refer to Oracle Corporation [90] for more information.

26.6 Miscellaneous Other Systems

An implementation of the engineering process model described in chapter 11 was developed by Anton Eygelaar [38] as part of his M.Sc. Eng. studies. The program is named PLEP. Refer to figure 26.1 for a screen shot of the user interface data entry screen.

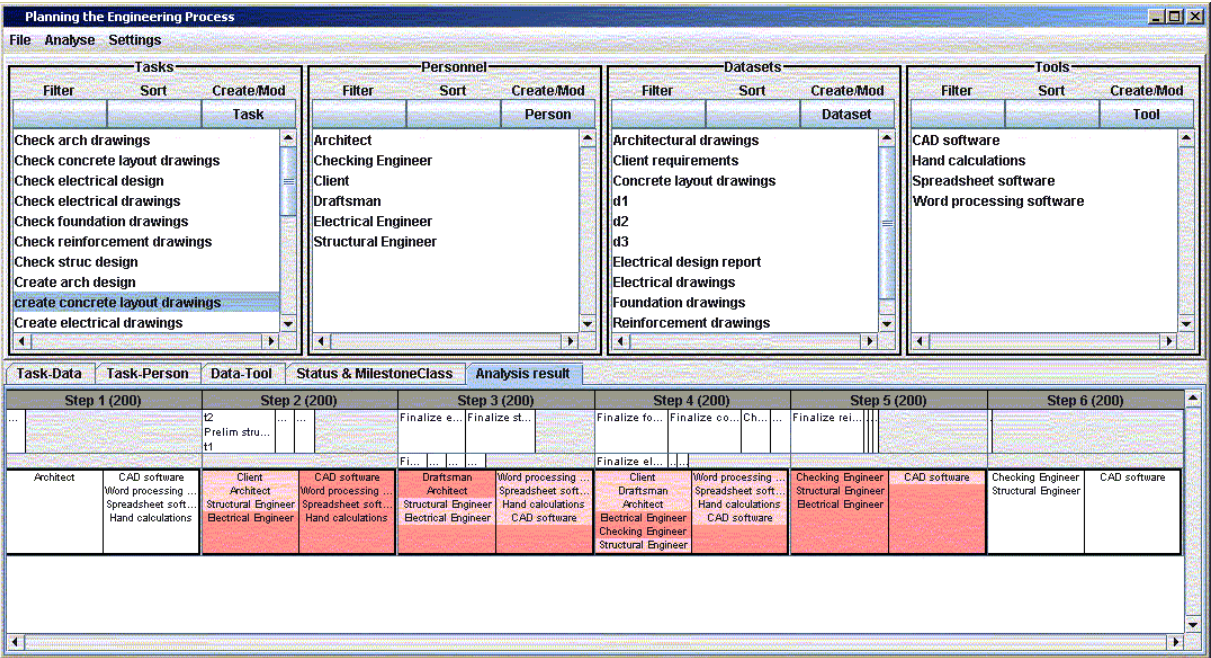


Figure 26.1: PLEP Engineering Process Software

Chapter 27

Conclusions on Addendum

The contents of this chapter summarises the conclusions contained in the Addendum.

Conclusions

1. Techniques developed in part II can be applied to model business functions and management systems for the business functions identified in part IV of this study.
2. The review of the professional service business from a functional viewpoint reported in part IV of this study indicates that the basic technology demonstrated in part II is suitable as the basis of basic enterprise systems, management reporting and decision support systems for engineering services enterprises.
3. The marketing related business objects and processes for a professional services business identified in chapter 17, the type of management reports shown in appendix L, as well as the theory and techniques available to construct management systems discussed in chapters 11, 12 and 13 can serve the marketing management function well with effective and efficient marketing systems and marketing management systems.
4. The review of the record keeping for costing and finance, bookkeeping and auditing functions of the professional services enterprise in chapter 18 indicates that the systems can be modelled and structured using basic mathematical constructs and database concepts such as those discussed in chapters 4 and 7.
5. Commercial accounting software systems can be linked to other business systems to provide generalised management reporting functionality structured as described in chapter 12. These business systems can be database driven.
6. The review of the personnel function of the professional services enterprise in chapter 19 indicates that personnel systems can be modelled and structured using basic mathematical and database concepts discussed in chapters 4 and 7.
7. Reported case studies of academic institutions, where the personnel function is supported by sophisticated workflow and management support and reporting systems, can be used as input for to develop systems for the professional services business.
8. Commercial personnel payroll software systems can be linked to other business systems to provide generalised management reporting functionality structured as described in chapter 12. These business systems can also be database driven.
9. The review of the production function of the professional services enterprise in chapter 20 indicates that production systems can be modelled and structured using basic concepts discussed in chapters 4 and 7.
10. The production function is one where the Engineering process model approach described in chapter 13 can be adopted with great success. Additional attributes can be added to the tasks, person, datasets and tool objects to model functionality required for production management. Links to other data contained in business databases, such as the personnel and project register databases, can be defined to support system functionality required.

11. Commercial project management software systems can be linked to other database driven business systems to provide generalised management reporting functionality structured as described in chapter 12.
12. The review of the facilities and document management function of the professional services enterprise in chapter 21 indicates that facilities and document management systems for the enterprise can be modelled and structured using basic concepts discussed in chapters 4 and 7.
13. The implementation of expensive commercial facilities and document management software systems is seen as an 'overkill' for the typical professional service provider. Facilities management does not constitute the main goal of the business.
14. A facilities management outsourcing service as a parallel product offered by an engineering consulting group of course requires the deployment of software to support the service offered.
15. Basic database driven facility and document systems can be developed to provide the required business process and management reporting functionality.
16. The review of the knowledge management function of the professional services enterprise in chapter 22 indicates that knowledge management systems for the enterprise can be modelled and structured using basic concepts discussed in chapters 4 and 7.
17. Basic database driven knowledge systems can be developed to provide the required business process and management reporting functionality.
18. The review of the logistics management function of the professional services enterprise in chapter 23 shows that logistics management systems for the enterprise can be modelled and structured using basic concepts discussed in chapters 4 and 7. Basic database driven systems can be developed to provide the required business logistics process and management reporting functionality.
19. Service businesses which provide specialised laboratory services need to pay special attention to the logistics function of that part of the business.
20. Basic database driven systems can be developed to provide the required administrative process and management reporting functionality not already available in other business systems or sub systems for professional services enterprises.
21. The review of the administration management function of the professional services enterprise in chapter 24 indicates that the Engineering process model is suited to support the project administration requirement for the core production process of the business.
22. As shown in chapter 25 risk management, with suitable systems deployed, is an important business function for the professional service business.
23. The PLEP Engineering process model program (Eygelaaar [38]) can be extended to link to the database environment for reporting functionality.
24. A high level business goal to develop, document and protect business production (project) knowledge as well as business procedure knowledge is required.

Part V

Appendices, Bibliography and References

Appendices

Appendix A

MATLAB implementation of set operations

The MATLAB code demonstrating set operations computed is set out below.

Contents

- set union , difference and intersection
- element contained in set, exclusive or set equality
- Relations and ordered pairs

```
%.....  
%  MATLAB Set Operations Example  
%.....  
clc  
clear all  
format compact
```

set union , difference and intersection

set union , difference and intersection

```
Set_Offices={'Pta_Office','Cpt_Office'}  
cellSetDisp(Set_Offices)  
Set_Departments={'Structures','Water Supply','Water Treatment'};  
cellSetDisp(Set_Departments)  
Set_New_Office={'Dbn_Office'}  
Set_All_Offices=union(Set_Offices,Set_New_Office)  
Set_New_Office_Diff=setdiff(Set_All_Offices,Set_New_Office)  
Set_Null={}  
DiffTest=setdiff(Set_Null,Set_Offices)  
Set_New_Department={'Offshore'}  
Set_All_Departments=union(Set_Departments,Set_New_Department)  
Set_Intersect=intersect(Set_All_Departments,Set_New_Department)
```

```
Set_Offices =  
    'Pta_Office'    'Cpt_Office'  
Set_Offices =  
{Pta_Office,Cpt_Office}  
Set_Departments =  
{Structures,Water Supply,Water Treatment}  
Set_New_Office =  
    'Dbn_Office'  
Set_All_Offices =  
    'Cpt_Office'    'Dbn_Office'    'Pta_Office'
```

```

Set_New_Office_Diff =
    'Cpt_Office'    'Pta_Office'
Set_Null =
    {}
DiffTest =
    {}
Set_New_Department =
    'Offshore'
Set_All_Departments =
    Columns 1 through 3
        'Offshore'    'Structures'    'Water Supply'
    Column 4
        'Water Treatment'
Set_Intersect =
    'Offshore'

```

element contined in set, exclusive or set equality

test member of set

```

Set1=Set_Offices
Element='Pta_Office'
disp(['Element in Set1 (1=true,0=false): '])
disp([ismember(Element,Set_Offices)])
% exclusive or
Set3=setxor(Set_All_Offices,Set_New_Office)
% test set equality
Set1=Set_Offices;
cellSetDisp(Set1)
Set2=Set_Offices;
cellSetDisp(Set2)
disp(['Set1=Set2 (1=true,0=false): '])
disp([(isequal(Set1,Set2))])
Set1=Set_Offices
Set2=Set_Departments
disp(['Set1=Set2 (1=true,0=false): '])
disp([(isequal(Set1,Set2))])
% powerset - not implemented

```

```

Set1 =
    'Pta_Office'    'Cpt_Office'
Element =
    Pta_Office
Element in Set1 (1=true,0=false):
     1
Set3 =
    'Cpt_Office'    'Pta_Office'
Set1 =
    {Pta_Office,Cpt_Office}
Set2 =
    {Pta_Office,Cpt_Office}
Set1=Set2 (1=true,0=false):
     1
Set1 =
    'Pta_Office'    'Cpt_Office'
Set2 =

```

```

    'Structures'      'Water Supply'      'Water Treatment'
Set1=Set2 (1=true,0=false):
    0

```

Relations and ordered pairs

Relations and ordered pairs

```

Office_Department=setCartesianProduct(Set_Offices,Set_Departments)
cellSetDisp(Office_Department)
All_Office_Department=setCartesianProduct(Set_All_Offices,Set_All_Departments);
cellSetDisp(All_Office_Department)
% set operations on ordered pairs not implemented
% DiffOrderedPairs=setdiff(Office_Department,All_Office_Department)
% find ordered pair in cartesian product
OrderPair1=All_Office_Department{4}
% disp([cellFind(All_Office_Department,0,OrderPair1)])
PairSet1=All_Office_Department;
PairSet2=All_Office_Department;
% equality of sets of ordered pairs and ordered pairs
disp(['PairSet1=PairSet2 (1=true,0=false): '])
disp([(cellCompare(PairSet1,PairSet2))])
disp([(cellCompare(PairSet1{6},PairSet2{6}))])
disp(PairSet1{5})
disp(PairSet1{6})
disp([(cellCompare(PairSet1{5},PairSet2{6}))])

```

```

Office_Department =
    {1x2 cell}    {1x2 cell}    {1x2 cell}
    {1x2 cell}    {1x2 cell}    {1x2 cell}
Office_Department =
{(Pta_Office,Structures),(Cpt_Office,Structures),(Pta_Office,Water Supply)
,(Cpt_Office,Water Supply),(Pta_Office,Water Treatment)
,(Cpt_Office,Water Treatment)}
All_Office_Department =
{(Cpt_Office,Offshore),(Dbn_Office,Offshore),(Pta_Office,Offshore)
,(Cpt_Office,Structures),(Dbn_Office,Structures),(Pta_Office,Structures)
,(Cpt_Office,Water Supply),(Dbn_Office,Water Supply)
,(Pta_Office,Water Supply),(Cpt_Office,Water Treatment)
,(Dbn_Office,Water Treatment),(Pta_Office,Water Treatment)}
OrderPair1 =
    'Cpt_Office'      'Structures'
PairSet1=PairSet2 (1=true,0=false):
    1
    1
    'Dbn_Office'      'Structures'
    'Pta_Office'      'Structures'
    0

```


Appendix B

MATLAB implementation of set functions

The MATLAB code listing for set functions with an example for the Cartesian product computation is given below.

Adding elements to sets

```
function [Aout] = addElementToSet(elem,A)
%.....
% Add element to set
%.....
% no additions to start with
Aout=A;
% Check if element is already in set
isInSet=0;
for i=1:size(A,2)
    if iscellstr(A)
        elemC= char(elem);
        AC= char(A(i));
        if size(elemC)== size(AC)
            if elemC == AC
                isInSet=1;
            end
        end
    else
        if elem == A(i)
            isInSet=1;
        end
    end
end
if isInSet==0
    Aout=[A,elem];
end
```

MATLAB Cell datastructure functionality for Sets

```
function [isequal]=cellCompare(cell1,cell2)
%.....
% compare MATLAB cell entries for equality
% isequal = 1 if
%.....
isequal=0;
%cell1
%cell2
nelem1=numel(cell1);
nelem2=numel(cell2);
if nelem1==nelem2
```

```

        % compare entry by entry
        isequal=1;
    for ielem=1:nelem1
        % compare only non cells - otherwise recurse
        %     iscell(cell1)
        %     iscell(cell2)
        if (iscell(cell1) & iscell(cell2))
            [isequalX]=cellCompare(cell1{ielem},cell2{ielem});
            disp(['isequal after recurse = ',num2str(isequalX)])
        %     disp(['is cell:',cell1])
        %     disp(['is cell:',cell2])
        isequal=isequalX;
        if(isequal==0) ; break; end
        end
        if ~((iscell(cell1) & iscell(cell2)))
        %     disp(['not cell:',cell1])
        %     disp(['not cell:',cell2])
        if ~(cell1(ielem)==cell2(ielem))
            isequal=0;
            break
        end
        end
    end
end
end
%disp(['isequal on exit = ',num2str(isequal)])

```

Locating strings in MATLAB cell datatype

```

function [ipos]=cellFind(cellArray,ipos,search_item)
%.....
% locate object in cell array
% ipos - position in cell array entry
%.....
ifind=0;
% search_item
%if iscell(search_item);
%   cellSetDisp(search_item,'search_item')
%end
% numel(cellArray)
% numel(search_item)
% loop over entries in cell array
search_itemX=search_item;
for icell=1:numel(cellArray)
    entry=cellArray{icell};
    if(ipos>0 & numel(cellArray{icell}) >0) ;
        entry=cellArray{icell}{ipos};
        if iscell(search_item)
            search_itemX=search_item{ipos};
        end
    end
    %     entry
    %     search_itemX
    [isequal]=cellCompare(entry,search_itemX);
    if [isequal]==1
        ifind=1;
        break
    end
end
ipos=[icell];
if(ifind==0); ipos=[]; end

```

MATLAB Cell content output in Set format including ordered pairs

```
function [] = cellSetDisp(c,s)
%.....
%   Display Set stored as cell in set format
%   c - cell array
%   s - string to be used as name for cell
%   Set output display of elements and ordered pairs
%.....
% check input arguments
error(nargchk(1,2,nargin));
% process only cell arrays
if ~iscell(c),
    error('MATLAB:celldisp:notCellArray', 'Must be a cell array.');
```

end

```
% isloose = strcmp(get(0,'formatspacing'),'loose');
lenline=50;
% set up Set name
if nargin==1, s = inputname(1); end
if isempty(s), s = 'ans'; end
% set name
disp([s ' = '])
% output Set elements buliding up output string
sizein=0;
strout='{';
[xs,sizeout]=size(strout);
for i=1:numel(c)
% output line if line length limit reached
    if sizeout > lenline; strout=soutput(strout); end
% add seperator , for entries after last
    if ~(i==1);
        strout=strcat(strout,',');
        [xs,sizeout]=size(strout);
    end
% process element as string, integer or real
    if ~iscell(c{i})
        [xs,sizein]=size(strout);
        strout=strcat(strout,convertCell(c{i}));
        [xs,sizeout]=size(strout);
    end
% process ordered pair or cell (higher level ordered pairs)
    if iscell(c{i})
        [xs,sizein]=size(strout);
        strout=cellOut(c{i},strout);
        [xs,sizeout]=size(strout);
    end
% remove seperator ,
    if (sizeout==sizein);
        strout=strout(1:sizeout-1);
    end
end
strout=strcat(strout,'}');
strout=soutput(strout);
%.....
function strout=cellOut(cellVal,strout)
% Output cell content as oredered pair strings
    nelement=numel(cellVal);
    if (nelement >0)
        strout=strcat(strout,'(');
        for ic=1:nelement
            strout=strcat(strout,convertCell(cellVal{ic}));
```

```

        if ic < nelement ; strout=strcat(strout,','); end
        end
        strout=strcat(strout,')');
        end
%.....
function sOut= convertCell(cellVal)
% convert cell contents to string
%   class(cellVal)
    sOut='';
    if iscell(cellVal)
        sOut=cell2str(cellVal);
    end
    if ischar(cellVal)
        sOut=char(cellVal);
    end
    if isfloat(cellVal)
        sOut=num2str(cellVal);
    end
    if isinteger(cellVal)
        sOut=int2str(cellVal);
    end
%.....
function strout=soutput(strout)
% output part of string
    [xs,sizein]=size(strout);
    if sizein > 0
        disp(strout)
        strout='';
    end

```

Output MATLAB Cell content to comma seperated file

```

function [] = CellWrite(csvFile,cellArray)
%.....
% Output cell format array to file row by row
% comma delimited format
%.....%
% loop over entries and output data
% list vertices
fid=fopen(csvFile,'w');
display (horzcat('File ',csvFile,' opened'))
nRow=size(cellArray,1);
nCol=size(cellArray,2);
for i=1:nRow
% build up row for output
    rowout=[];
    for j=1:nCol
        rowout=[rowout,char(cellArray(i,j))];
        if j<nCol
            rowout=[rowout,','];
        end
    end
    fprintf(fid,'%s \n',rowout);
end
% close output file
fstatus=fclose(fid);
display (horzcat('File ',csvFile,' closed'))

```

Set cartesian product in cell array format

```

function [C] = setCartesianProduct(A,B)
%.....
% Set up set Cartesian product in cell array format
%.....
C={};
%disp(strcat('numel(A)= ', num2str(numel(A))))
%disp(strcat('numel(B)= ', num2str(numel(B))))
for i=1:numel(A)
    for j=1:numel(B)
        C{i,j}=[A(i),B(j)];
    end
end
end

```

Set up a relation in ordered pair format by selecting entries from set cartesian product using boolean matrix relation format as input

```

function [R] = setSubSetCartesianProduct(A,B,R_AB )
%.....
% Set up subset R of Cartesian product of matrices A and B
% R_AB is a boolean matrix which selects the elements of
% A x B which needs to be included in R
%.....
R={};
%disp(strcat('numel(A)= ', num2str(numel(A))))
%disp(strcat('numel(B)= ', num2str(numel(B))))
%numel(B)
for i=1:numel(A)
    for j=1:numel(B)
        R{i,j}=[];
        if(R_AB(i,j))
            R{i,j}=[A(i),B(j)];
        end
    end
end
end
end

```

```

%.....
% Test Set Cartesian prod - up to level 2
%.....
format compact
clear all
clc
% Single entry matrices - Cartesian product
A={'1'}
B={'a'}
S_Union=union(A,B)
prod=setCartesianProduct(A,B)
class(prod)
celldisp(prod)
cellSetDisp(prod,'prod')
% Mutiple entry matrices - Cartesian product
A={'1','2','3'}
B={'a','b'}
S_Union=union(A,B)
prod=setCartesianProduct(A,B)

```

```

class(prod)
celldisp(prod)
cellSetDisp(prod,'prod')
% Multiple Cartesian product
C={'x','y'}
% C={'x','y','z'}
prod2=setCartesianProduct(prod,C)
celldisp(prod2)
cellSetDisp(prod2,'prod2')

```

```

A =
    '1'
B =
    'a'
S_Union =
    '1'    'a'
prod =
    {1x2 cell}
ans =
    cell
prod{1}{1} =
    1
prod{1}{2} =
    a
prod =
    {(1,a)}
A =
    '1'    '2'    '3'
B =
    'a'    'b'
S_Union =
    '1'    '2'    '3'    'a'    'b'
prod =
    {1x2 cell}    {1x2 cell}
    {1x2 cell}    {1x2 cell}
    {1x2 cell}    {1x2 cell}
ans =
    cell
prod{1,1}{1} =
    1
prod{1,1}{2} =
    a
prod{2,1}{1} =
    2
prod{2,1}{2} =
    a
prod{3,1}{1} =
    3
prod{3,1}{2} =
    a
prod{1,2}{1} =
    1
prod{1,2}{2} =
    b
prod{2,2}{1} =
    2
prod{2,2}{2} =
    b
prod{3,2}{1} =

```

```

3
prod{3,2}{2} =
b
prod =
{(1,a),(2,a),(3,a),(1,b),(2,b),(3,b)}
C =
      'x'      'y'
prod2 =
      {1x2 cell}      {1x2 cell}
      {1x2 cell}      {1x2 cell}
      {1x2 cell}      {1x2 cell}
      {1x2 cell}      {1x2 cell}
      {1x2 cell}      {1x2 cell}
      {1x2 cell}      {1x2 cell}
prod2{1,1}{1}{1} =
1
prod2{1,1}{1}{2} =
a
prod2{1,1}{2} =
x
prod2{2,1}{1}{1} =
2
prod2{2,1}{1}{2} =
a
prod2{2,1}{2} =
x
prod2{3,1}{1}{1} =
3
prod2{3,1}{1}{2} =
a
prod2{3,1}{2} =
x
prod2{4,1}{1}{1} =
1
prod2{4,1}{1}{2} =
b
prod2{4,1}{2} =
x
prod2{5,1}{1}{1} =
2
prod2{5,1}{1}{2} =
b
prod2{5,1}{2} =
x
prod2{6,1}{1}{1} =
3
prod2{6,1}{1}{2} =
b
prod2{6,1}{2} =
x
prod2{1,2}{1}{1} =
1
prod2{1,2}{1}{2} =
a
prod2{1,2}{2} =
y
prod2{2,2}{1}{1} =
2
prod2{2,2}{1}{2} =
a
prod2{2,2}{2} =

```

```

y
prod2{3,2}{1}{1} =
3
prod2{3,2}{1}{2} =
a
prod2{3,2}{2} =
y
prod2{4,2}{1}{1} =
1
prod2{4,2}{1}{2} =
b
prod2{4,2}{2} =
y
prod2{5,2}{1}{1} =
2
prod2{5,2}{1}{2} =
b
prod2{5,2}{2} =
y
prod2{6,2}{1}{1} =
3
prod2{6,2}{1}{2} =
b
prod2{6,2}{2} =
y
prod2 =
{((1,a),x),((2,a),x),((3,a),x),((1,b),x),((2,b),x),((3,b),x)
,((1,a),y),((2,a),y),((3,a),y),((1,b),y),((2,b),y),((3,b),y)}

```


Appendix C

MATLAB implementation of relation operations

C.1 Relation operations programmed in MATLAB

The MATLAB code with functions required for relation operations computed is set out below.

MATLAB functionality for relational algebra using boolean matrix representation of relations

```
%.....  
%  
%   Relational Algebra - Boolean matrix representation  
%   MATLAB Matrix operations  
%.....  
display(strcat('MATLAB implementation of relational algebra', ...  
               ' boolean matrix operations in inline functions'))  
%.....  
% zero, one and identity  
zeroB=inline('logical(zeros(n,m))','n','m'); % e.g. Z4=zeroB(4,4)  
oneB=inline('logical(ones(n,m))','n','m'); % e.g. Z4=zeroB(4,4)  
identityB=inline('logical(eye(n))','n'); % e.g. I3=identityB(3)  
%.....  
% Basic relational operations  
productR= ...  
inline('logical(mod(ones(size(fix(x)*fix(y))), (fix(x)*fix(y))+1))','x','y');  
% Note: MATLAB standard set union is a=union([1,1],[1,1])  
unionR= ...  
    inline('logical(mod(ones(size(x)), (x+y)+1))','x','y') ;  
intersectionR= ...  
    inline('logical(mod(ones(size(x)), (x.*y)+1))','x','y');  
differenceR=inline( ...  
    'logical(mod(ones(size(x)), (x-mod(ones(size(x)), (x.*y)+1))+1))','x','y');  
complementR=inline('logical(zeros(size(x))+not(logical(x)))','x');  
transposeR=inline('transpose(x)','x');  
%.....
```

Testing relations in Boolean format for equality

```
function isEqual = isRelEqual(R,S)  
%.....  
% Test relations R and S in boolean matrix form for equality  
%.....  
isEqual=0;  
nRowsR=size(R,1);  
nColumnsR=size(R,2);
```

```

nRowsS=size(S,1);
nColumnsS=size(S,2);
if (nRowsR == nRowsS) & (nColumnsR == nColumnsS)
%   isEqual=1;
%   i=1;
%   while (i<=nRowsR) & (isEqual==1)
%       j=1;
%       while (j<=nColumnsR) & (isEqual==1)
%           if R(i,j)~=S(i,j)   isEqual=0;
%               end
%           j=j+1;
%       end
%       i=i+1;
%   end
isEqual=all(all((R==S)==1)==1);
end

```

Boolean product of boolean entry and string yielding string for logical entry = 1 or blank string for logical entry = 0

```

function [cellOut]=productboolString(cellIn,bool)
%.....
%   string multiply boolean
%   bool = 1 cell Array output
%   bool = 0 blank output
%.....
cellOut={' '};
if(bool==1)
    cellOut=cellIn;
end

function [cellOut]=stringMultBool(cellIn,boolean)
%.....
%   string multiply boolean
%   boolean element = 1 concatenate
%   boolean element = 0 blank output
%.....
nrBool=size(boolean,1);
nCBool=size(boolean,2);
nrCell=size(cellIn,1);
nCCell=size(cellIn,2);
if (nCCell==nrBool)
    for ir=1:nRCell
        for ic=1:nCBool
            cellOut(ir,ic)={' '};
        end
    end
    for ir=1:nRCell
        for ic=1:nCBool
            for ik=1:nCCell
                cellOut(ir,ic)=strcat(cellOut(ir,ic), ...
                    productboolString(cellIn{ir,ik}, ...
                        boolean(ik,ic)));
            end
        end
    end
else
    disp('Error CellIn and boolean columns / rows incompatibile')
end

```

Appendix D

MATLAB implementation of graph operations

The MATLAB code demonstrating graph operations computed is set out below.

Generate graph adjacency list given graph adjacency matrix

```
function [AdjList,nrowL,ncolL] = AdjacencyList(AdjMatrix)
%.....
% Generate adjacency list given adjacency matrix
% Adjacency list in matrix format - ignore 0 entries
%.....
AdjList=[];
nrowA=size(AdjMatrix,1);
ncolA=size(AdjMatrix,2);
for ir=1:nrowA
    irowL=ir;
    AdjList(irowL,1)=ir;
    icoll=1;
    for ic=1:ncolA
        if(AdjMatrix(ir,ic)==1)
            icoll=icoll+1;
            AdjList(irowL,icoll)=ic;
        end
    end
end
nrowL=size(AdjList,1);
ncolL=size(AdjList,2);
```

Extract all graph edges linked to a vertex and return the subgraph in adjacency matrix format

```
function [adjMatrix,vactive] = adjListExtract(adjList,nvertex)
%.....
% Extract all edges linked to a vertex and return
% adjMatrix - subgraph in adjacency matrix format
% vactive - Boolean list of active vertices
% can be converted to list format if required
% nvertex - vertex number to process
%.....
% initialise output
adjMatrix=[];
nVertices=max(size(adjList,1),max(max(adjList)));
if not((nvertex>nVertices))
% set up blank adjacency matrix
adjMatrixInterim=zeros(nVertices);
vactive=zeros(1,nVertices);
% set limits to active vertex numbers
for nv=1:size(adjList,1)
```

```

        nEdgesRow=size(adjList(nv,:),2);
        for ne=2:nEdgesRow
% end vertex of edge
            mv=adjList(nv,ne);
% skip zero entries as well as entries not linked to selected vertex
            if not(mv==0)&& (mv==nvertex)
                adjMatrixInterim(nv,mv)=1;
% keep track of active vertex entries
                vactive(nv)=1;
                vactive(mv)=1;
            end
        end
    end
end
% disp(['vactive: ',num2str(vactive)])
% adjMatrixInterim
% Add active columns to output matrix
    adjCols=[];
    for iv=1:nVertices
        if (vactive(iv)==1)
            adjCols=[adjCols,adjMatrixInterim(:,iv)];
        end
    end
end
% Add active rows to output matrix
% adjCols
for iv=1:nVertices
    if (vactive(iv)==1)
        adjMatrix=[adjMatrix;adjCols(iv,:)];
    end
end
end
end

```

Convert graph data representation from adjacency List to adjacency Matrix format

```

function [adjMatrix]=adjMatrixFromList(adjList)
%.....
% Convert graph representation from adjacency List to adjacency Matrix
%.....
% maximum entry in adjacency list is largest vertex number
nVertices=max(size(adjList,1),max(max(adjList)));
% set up blank adjacency matrix
adjMatrix=zeros(nVertices);
% adjMatrix=[];
% Loop over vertices and make entries in adjacency matrix
for nv=1:size(adjList,1)
    nEdgesRow=size(adjList(nv,:),2);
    for ne=2:nEdgesRow
% end vertex of edge
        mv=adjList(nv,ne);
% skip zero entries
        if not(mv==0)
            adjMatrix(nv,mv)=1;
        end
    end
end
end
end

```

Generate adjacency list of all outgoing edges of a given graph vertex

```

function [Edges,nedge] = ALOutgoingEdges(AdjList,vertex)
%.....
% Generate list of edges
% Adjacency list in matrix format - ignore 0 entries
%.....
Edges=[];
nrowA=size(AdjList,1);
ncolA=size(AdjList,2);
nedge=0;
% vertex not in graph
if(vertex>0) & (vertex<=nrowA)
for icol=2:ncolA
    if(AdjList(vertex,icol)>0)
        nedge=nedge+1;
        Edges=[Edges,[vertex,AdjList(vertex,icol)]];
    end
end
end
end

```

Generate list of graph vertices with no incoming edges given adjacency list

```

function [Vertices,nvert] = ALVertWithNoIncEdges(AdjList)
%.....
% Generate list of vertices with no incoming edges
% given adjacency list
%.....
Vertices=[];
nvert=0;
nrowL=size(AdjList,1);
ncolL=size(AdjList,2);
for irow=1:nrowL
[r,c]=find(AdjList(:,2:ncolL)==irow)
iverts=size(r,1)
    if(iverts==0)
        Vertices=[Vertices,irow]
        nvert=nvert+1
    end
end
end

```

Apply depth first search [DFS] algorithm to graph given in adjacency matrix format

```

function [tree,dfsArray] = DepthFirstSearch(R)
    global List mark ipos dfsArray
%.....
%   DFS of graph
%   Input: R - graph in Adjacency matrix format
%   Output dfsArray - traversal of indices in DFS order
%   tree - Graph spanning tree in adjacency matrix format
%.....
RelationalAlgebraBoolean
nVertex=size(R,1);
[adjList,nrowL,ncolL] = AdjacencyListM(R)
dfsArray=[];
% tree=[];
tree=zeros(size(R));
List={};
ipos=0;

```

```

% celldisp(List)
for v=1:nVertex
    mark(v)=0;
end
while (any(mark==0)==1)
    x=min(find(mark==0));
    % disp(['into search x mark: ',num2str(mark)])
    % [List,mark]=search(x,adjList,nrowL,ncolL,mark,List)
    search(x,adjList,nrowL,ncolL);
    % disp(['outof search x mark: ',num2str(mark)])
    while (size(List,2)>0)
        % display(['Size of list = ',num2str(size(List))])
        L=List{end};
        v=L(1);
        w=L(2);
        List=List(1:end-1);
        if(mark(w)==0);
            tree(v,w)=1;
        % disp(['into search v mark: ',num2str(mark)])
        % [List,mark]=search(v,adjList,nrowL,ncolL,mark,List)
        search(w,adjList,nrowL,ncolL);
        % disp(['outof search v mark: ',num2str(mark)])
        % disp(['outof search size List: ',num2str(size(List,2))])
    end
end
end

function []=search(v,adjList,nrowL,ncolL)
%.....
% Updates List to include next edge in graph
% Mark the vertex as processed
% Update the processing sequence array dfsArray
% by adding the vertex processed to it
%.....
global List mark ipos dfsArray
dfsArray=[dfsArray,v];
mark(v)=1;
for iv=2:ncolL
    iw=adjList(v,iv);
    if(iw>0)
        List=[List,[v,iw]];
    % disp ([num2str(v),' ',num2str(iw),' Added to List'])
    % celldisp(List);
    end
end
end

```

Determine indegrees of graph vertices with graph given in adjacency matrix format

```

function [InDegree] = inDegreeG(V)
%.....
% Determine indegrees of graph vertices - graph given in
% adjacency matrix format - column entry sums in vector form
%.....
RelationalAlgebraBoolean;
C1=ones(size(V,2),1);
InDegree=V*C1;

```

Determine outdegrees of graph vertices with graph given in adjacency matrix format

```

function [OutDegree] = outDegreeG(V)
%.....
% Determine outdegrees of graph vertices - graph given in
% adjacency matrix format - row entry sums in vector form
%.....
RelationalAlgebraBoolean;
R1=ones(1,size(V,2));
OutDegree=R1*V;

```

Extract set sub adjacency matrices form topological sort matrix given topol sort adjacency matrix as well as original tree structure adjacency matrix

```

function [subAdj,subVactiveCol,subVactiveRow] ...
    = SubAdjMatricesExTopolSort(TreeAdj,TOut)
%.....
% Extract set sub adjacency matrices form topological sort matrix
% given topol sort adjacency matrix as well as original tree
% structure adjacency matrix
% Input:
% TreeAdj - Tree adjacency matrix
% TOut - Topological sorting graph adjacency matrix
% Output:
% subAdj - Cell array of sub adjacency matrices
% subVactiveCol - Cell array of Boolean vectors of active vertices
%                included in subAdj - columns
% subVactiveRow - Cell array of Boolean vectors of active vertices
%                included in subAdj - rows
%.....
isubMat=0;
% size of topological sort adjacency matrix
% rows - vertices
% columns - sort levels / classes
nRow=size(TOut,1)
nCol=size(TOut,2)
% Loop over columns in topological sort graph adjacency matrix
for j=1:nCol
% list of vertices column list stores for extractions
collist=[];
% boolean array of active vertices in position - columns
vactiveCol=[];
% Loop over rows in topological sort adjacency matrix
    for i=1:nRow
        if(TOut(i,j)~= 0)
% add vertex to column list
            collist=[collist,i];
            vactiveCol(i)=1;
        end
    end
% Obtain set of active vertices for level op topological sort
    collist=unique(collist);
% Extract columns listed (if any)
    subMat=[];
    kCol=size(collist,2);
    if kCol > 0
        for k=1:kCol
            subMat=[subMat,TreeAdj(:,collist(k))] ;
        end
    end
end

```

```

% subMat
% number of rows in submatrix
lRow=size(subMat,1);
% mark all vertices active for rows
vactiveRow=ones(lRow,1);
for lr=1:lRow
    if sum(subMat(lr,:))==0;
        vactiveRow(lr)=0;
    end
end
% remove zero-rows from matrix subMat
lr=1;
while lr<=lRow
    if sum(subMat(lr,:))==0
        subMat(lr,:)=[];
        lRow=lRow-1;
        lr=0;
    end
    lr=lr+1;
end % while
if sum(subMat)==0
    subMat=[];
end
% Store derived adjacency matrix in cell array
% subMat
% vactiveRow
% subAdj={} % - not correct
if(sum(size(subMat)))>0
    isubMat=isubMat+1;
    subAdj{isubMat}=subMat;
    subVactiveCol{isubMat}=vactiveCol;
    subVactiveRow{isubMat}=vactiveRow;
end
% next column
end % loop over topological sort adj matrix columns

```

Read graph data from .tgf format file for use with yEd graph display

```

function [vertexLabels,edgeLabels,R] = TgfRead(tgfFile)
%.....
%   tgfFile: File for input of data
%   vertexLabels: cell array with vertex label strings
%   edgeLabels: cell array with edge label strings
%   R adjacency matrix
%.....
% Input / Read .tgf file for relation for plotting with yEd
% Sample file
% 1 0
% 2 1
% 3 2
% 4 A
% 5 B
% 6 C
% 7 D
% 8 E
% #
% 2 1 Edge10
% 3 1 Edge20
% 4 2 EdgeA1
% 5 2 EdgeB1

```



```

% 6 2 EdgeC1
% 7 3 EdgeD2
% 8 3 EdgeE2
%.....
% open file
fid=fopen(tgfFile,'r');
display (horzcat('File ',tgfFile,' opened'))
lineInputV=' ';
% vertices
vertexLabels={};
numVertices=0;
while not(lineInputV(1,1)=='#')
    lineInputV=fgetL(fid);
    if lineInputV(1,1)=='#'
        break
    end
    numVertices=numVertices+1;
    V=textscan(lineInputV,'%d %s','delimiter', '\n');
%    celldisp(V)
    vertexLabels(numVertices)={' '};
    if(size(V{2},1) > 0)
        vertexLabels(numVertices)=V{2};
    end
end
% edges labels & adjacency matrix
edgeLabelsIn={};
% R=logical([]);
% adjacency matrix to be square
R=logical(zeros(numVertices));
lineInputE=' ';
Rrows=0;
Rcols=0;
while not(lineInputE=='-1')
    lineInputE=fgetL(fid);
    if lineInputE=='-1'
        break
    end
    E=textscan(lineInputE,'%d %d %s','delimiter', '\n');
    vertex1=double(E{1});
    if(vertex1>Rrows) Rrows=vertex1;end
    vertex2=double(E{2});
    if(vertex2>Rcols) Rcols=vertex2;end
    R(vertex1,vertex2)=1;
    edgeLabelsIn(vertex1,vertex2)={' '};
    if(size(E{3},1) > 0)
        edgeLabelsIn(vertex1,vertex2)=E{3};
    end
end
end
%Rrows
%Rcols
%sum(sum(R))
% store edgelabels in sequence of edges define in R
% row by row
edgeLabels={};
numEdges=0;
for ir=1:Rrows
    for ic=1:Rcols
        if(R(ir,ic)==1)
            numEdges=numEdges+1;
            edgeLabels(numEdges)=edgeLabelsIn(ir,ic);
        end
    end
end

```

```

        end
    end
    %size(R)
    %size(edgeLabels)
    % close output file
    fstatus=fclose(fid);
    display (horzcat('File ',tgfFile,' closed'))

```

Write graph data to .tgf format file for use with yEd graph display

```

function [] = TgfWrite(tgfFile,R,isHomog,vertexLabels,edgeLabels)
%.....
% Output relation data file in directed graph data .tgf file format for use in
% yEd/yFiles software
% tgfFile: file name / Character string
% isHomog(eneous): = 1 or 0 relation between one or two sets / bipartite graph
% R: Adjacency / boolean relation matrix / double array
% vertexLabels: vertex labels for plot / String cell array - required
% edgeLabels: edge labels for plot / String cell array - optional
%.....
% sample filename:
% tgfFile='TestFileA.tgf'
% Output .tgf file for relation for plotting with yEd
% Sample file contents:
% 1 0
% 2 1
% 3 2
% 4 A
% 5 B
% 6 C
% 7 D
% 8 E
% #
% 2 1 Edge10
% 3 1 Edge20
% 4 2 EdgeA1
% 5 2 EdgeB1
% 6 2 EdgeC1
% 7 3 EdgeD2
% 8 3 EdgeE2
% Sample vertexLabels:
% vertexLabels={'0','1','2','A','B','C','D','E'}
% Sample edgelabels:
% edgeLabels={'Edge10', ...
% 'Edge20', ...
% 'EdgeA1', ...
% 'EdgeB1', ...
% 'EdgeC1', ...
% 'EdgeD2', ...
% 'EdgeE2'}
% Sample relation boolean matrix / adjacency matrix for directed graph:
% R=logical([0 1 1 0 0 0 0 0; ...
%   0 0 0 1 1 1 0 0; ...
%   0 0 0 0 0 0 1 1; ...
%   0 0 0 0 1 1 0 0; ...
%   0 0 1 1 0 0 0 0; ...
%   1 1 0 0 0 0 0 0; ...
%   1 0 0 0 1 1 1 1])
%.....%
% loop over entries and output data

```

```

% list vertices
fid=fopen(tgfFile,'w');
display (horzcat('File ',tgfFile,' opened'))
for i=1:size(vertexLabels,2)
    % output only non-zero length labels with
    if size( vertexLabels{i},2)>0
        fprintf(fid,'%u %s \n',i,vertexLabels{i});
    end
end
fprintf(fid,'%s \n','');
% homogeneous relation
% loop over relation boolean matrix to output edges
iedge=0;
nRows=size(R,1)
nCols=size(R,2)
for j=1:nRows
    for k=1:nCols
        if(R(j,k) == true )
            iedge=iedge+1;
            label=' ';
            if(iedge <= size(edgeLabels,2));
                label=edgeLabels{iedge};
            end
            jpos=j;
            kpos=k;
            if(not(isHomog==1))
                kpos=k+nRows;
            end
            fprintf(fid,'%u %u %s \n',jpos,kpos,label);
        end
    end
end
% close output file
fstatus=fclose(fid);
display (horzcat('File ',tgfFile,' closed'))

```

Graph topological sort algorithm for Directed Acyclic Graph (DAG)

```

function [TaskSchedule] = TopolSort(T)
%.....
% Determine task schedule from task successor / predecessor matrix
% Using topological sort algorithm on DAG
% DAG - Directed Acyclic Graph
%.....
RelationalAlgebraBoolean
T
nVertex=size(T,1)
v{1}=logical(ones(nVertex,1));
for i=1:nVertex
    % Transpose incidence relation in matrix form
    v{i+1}=productR(T',v{i});
    if(sum(v{i+1})==0)
        ncol=i+1;
        break;
    end
end
TaskSchedule=[];
for i=1:ncol-1
    TaskSchedule=[TaskSchedule, ...
        intersectionR(logical(v{i}),not(logical(v{i+1})))];
end

```

```
end
```

Graph topological breadth first search

```
function [TaskSchedule] = TopolSortBFSTasks(T)
%.....
% Determine task schedule from task successor / predecessor matrix
%.....
T
A=[];
B=[];
idim=size(T,1);
for i=1:idim
    C(i)=i;
end
% prepare task schedule blank column to add on
TaskSchedule=[];
level=1;
for j=1:idim
    TaskColumn(j)=0;
end
TaskColumn=TaskColumn';
% Find vertex without predecessor
for i=1:idim
    tstart=0;
    if sum(T(i:i)) == 0 tstart=i;
        break
    end
end
if tstart==0
    disp '** Error ** No task without predecessor found'
    exit
end
% disp (['tstart = ',num2str(tstart)]);
A=[A,tstart];
% while size(A,2) > 0
CHasElements=1;
while (CHasElements == 1)
    disp '***** while loop begin *****';
    A;
    B;
    C;
    % if A is empty break
    if size(A,2)==0
        CHasElements = 1
        break
    end
    % Add all successors of elements in A to B
    for j=1:size(A,2)
        for i=1:idim
            if (T(A(j),i)) ~= 0
                [B] = addElementToSet(i,B);
            end
        end
    end
    disp 'After successor determination';
    A;
    B;
    C;
```

```

    isizeA=size(A,2);
    isizeB=size(B,2);
    % Remove all elements from A which are in B
    for i=1:isizeB
        for j=1:isizeA
            if j > size(A,2)
                break
            end
            if (A(j)==B(i))
                A(j)=[];
            end
        end
    end
    isizeA=size(A,2);
    end
    disp 'After removals from A and B ++++++';
    A;
    B;
    % Remove all elements from C which are in A
    isizeA=size(A,2);
    isizeC=size(C,2);
    for i=1:isizeA
        for j=1:isizeC
            if j > size(C,2)
                break
            end
            if (A(i)==C(j))
                C(j)=[];
            end
        end
    end
    isizeC=size(C,2);
    end
    end
    disp 'After removals from A and C *****';
    A;
    B;
    C;
    TaskSchedule=[TaskSchedule,TaskColumn];
    for i=1:size(A,2)
        TaskSchedule(A(i),level)=1;
    end
    TaskSchedule
    level=level+1
    % Swop A and B
    A=B;
    % clear B
    B=[];
    if size(C,2)==0
        CHasElements=0
    end
    % for testing
    % pause
    end

```

Appendix E

MATLAB System Function Example

The MATLAB code demonstrating the system function example is set out below.

Contents

- MATLAB Relational algebra functionality
- System inputs and outputs
- System state variables and discrete time steps
- Input and output relation domains
- Define input trajectory f as boolean format relation
- Relation to link system states to system output
- Next state mapping for system
- Time step display
- System initialisation
- System operations looping over time steps
- System output trajectory

```
%.....  
% System function example  
%.....  
clc  
clear all  
format compact
```

MATLAB Relational algebra functionality

Set up inline function availability

```
RelationalAlgebraBoolean
```

```
MATLAB implementation of relational algebra boolean matrix operations in inline functions
```

System inputs and outputs

System inputs

```
I_s={'setOn','setOff'};  
cellSetDisp(I_s,'I_s')  
% celldisp(I_s,'I_s')  
% System Outputs  
O_s={'lightOn','lightOff'};  
cellSetDisp(O_s,'O_s')
```

```
I_s =
{setOn,setOff}
O_s =
{lightOn,lightOff}
```

System state variables and discrete time steps

System state variables - switch status

```
S_s={'on','off'};
cellSetDisp(S_s,'S_s')
% system counter / timer set
T={0,1,2,3,4,5,6};
cellSetDisp(T,'T')
```

```
S_s =
{on,off}
T =
{0,1,2,3,4,5,6}
```

Input and output relation domains

input relation domain

```
R_in_domain=setCartesianProduct(I_s,S_s);
cellSetDisp(R_in_domain,'R_in_domain')
% cellDisp(R_in_domain)
% output relation domain
R_out_domain=setCartesianProduct(S_s,O_s);
cellSetDisp(R_out_domain,'R_out_domain')
% cellDisp(R_out_domain)
```

```
R_in_domain =
{(setOn,on),(setOff,on),(setOn,off),(setOff,off)}
R_out_domain =
{(on,lightOn),(off,lightOn),(on,lightOff),(off,lightOff)}
```

Define input trajectory f as boolean format relation

input trajectory relation on $T \times I_s$

```
R_TI_s=logical([1 0 ; ...
                0 1 ; ...
                1 0 ; ...
                1 0 ; ...
                0 1 ; ...
```

```

        0 1 ; ...
        1 0])
f=setSubsetCartesianProduct(T,I_s,R_TI_s);
% cellDisp(f)
cellSetDisp(f,'f')

```

```

R_TI_s =
     1     0
     0     1
     1     0
     1     0
     0     1
     0     1
     1     0

f =
{(0,setOn),(2,setOn),(3,setOn),(6,setOn),(1,setOff)
,(4,setOff),(5,setOff)}

```

Relation to link system states to system output

```

R_s_bool=logical([1 0 ; ...
                  0 1 ])
R_s=setSubsetCartesianProduct(S_s,O_s,R_s_bool)
R_Ss_Is_Ss=logical([1 0 ; ...
                    0 1 ; ...
                    1 0 ; ...
                    0 1 ])
X_s=productR(R_Ss_Is_Ss,R_s_bool)

```

```

R_s_bool =
     1     0
     0     1

R_s =
{1x2 cell}      []
      []      {1x2 cell}

R_Ss_Is_Ss =
     1     0
     0     1
     1     0
     0     1

X_s =
     1     0
     0     1
     1     0
     0     1

```

Next state mapping for system

nextstate mapping domain


```

N_s_domain=setCartesianProduct(R_in_domain,S_s);
cellSetDisp(N_s_domain,'N_s_domain')
% select elements of nextstate mapping
N_s_Select=logical([1 0 ; ...
                   0 1 ; ...
                   1 0 ; ...
                   0 1 ])
N_s=setSubsetCartesianProduct(R_in_domain,S_s,N_s_Select);
cellSetDisp(N_s,'N_s')

```

```

N_s_domain =
{((setOn,on),on),((setOff,on),on),((setOn,off),on),((setOff,off),on)
,((setOn,on),off),((setOff,on),off),((setOn,off),off)
,((setOff,off),off)}
N_s_Select =
    1     0
    0     1
    1     0
    0     1
N_s =
{((setOn,on),on),((setOn,off),on),((setOff,on),off)
,((setOff,off),off)}

```

Time step display

timestep loop display

```

for it=1:size(T,2)
    [ipos]=cellFind(f,1,T{it});
    icol=ipos(1);
    disp ([num2str(it),' ',num2str(T{it}),' ',f{icol}{2}])
end

```

```

1 0 setOn
2 1 setOff
3 2 setOn
4 3 setOn
5 4 setOff
6 5 setOff
7 6 setOn

```

System initialisation

system start start

```

S_start='on'
A{1}=S_start;
cellSetDisp(A)

```

```
S_start =
on
A =
{on}
```

System operations looping over time steps

loop over time steps extract input influence / action extract next state and determine output

```
for it=1:size(T,2)
% for it=1:2
    disp(['***** it=',num2str(it)])
    [ipos]=cellFind(f,1,T{it});
    icol=ipos(1);
    disp ([num2str(it),' ',num2str(T{it}),' ',f{icol}{2}])
% build ordered pair with active state and input
    B{1}=f{icol}{2};
    select= setCartesianProduct(B,A);
% select{1}
    cellSetDisp(select,'select')
% find next state in next state set
    [jpos]=cellFind(N_s,1,select);
% next state
    next_state=N_s{jpos}{2};
    A{1}=next_state;
    cellSetDisp(A);
% find output attribute setting
    output=cellFind(R_s,1,next_state);
% build output trajectory
    t_out{it}=R_s{output}{2};
end
```

```
***** it=1
1 0 setOn
select =
{(setOn,on)}
A =
{on}
***** it=2
2 1 setOff
select =
{(setOff,on)}
A =
{off}
***** it=3
3 2 setOn
select =
{(setOn,off)}
A =
{on}
***** it=4
4 3 setOn
select =
{(setOn,on)}
A =
```

```
{on}
***** it=5
5 4 setOff
select =
{(setOff,on)}
A =
{off}
***** it=6
6 5 setOff
select =
{(setOff,off)}
A =
{off}
***** it=7
7 6 setOn
select =
{(setOn,off)}
A =
{on}
```

System output trajectory

```
disp('System output trajectory:')
cellSetDisp(t_out)
```

```
System output trajectory:
t_out =
{lightOn,lightOff,lightOn,lightOn,lightOff,lightOff
,lightOn}
```

Appendix F

MATLAB Literal String Processing Functionality

Literal String Processing MATLAB Functionality.

```
function [cellOut]=stringMultBool(cellIn,boolean)
%.....
%   string multiply boolean
%   boolean element = 1 concatenate
%   boolean element = 0 blank output
%.....
nRBool=size(boolean,1);
nCBool=size(boolean,2);
nRCell=size(cellIn,1);
nCCell=size(cellIn,2);
if (nCCell==nRBool)
    for ir=1:nRCell
        for ic=1:nCBool
            cellOut(ir,ic)={' '};
        end
    end
    for ir=1:nRCell
        for ic=1:nCBool
            for ik=1:nCCell
                cellOut(ir,ic)=strcat(cellOut(ir,ic), ...
                    productboolString(cellIn{ir,ik}, ...
                        boolean(ik,ic)));
            end
        end
    end
else
    disp('Error CellIn and boolean columns / rows incompatable')
end
```

```
function [cellOut]=productboolString(cellIn,bool)
%.....
%   string multiply boolean
%   bool = 1 cell Array output
%   bool = 0 blank output
%.....
cellOut={' '};
if(bool==1)
    cellOut=cellIn;
end
```

```

%.....
% String concatenation test
%.....
clc
clear all
format compact
A={'a11' , 'a12r'; ...
  'a21' , 'a22e'}
B={'b11' , 'b12r'; ...
  'b21' , 'b22e'}
x=strcat(A)
y=strcat(A,B)
b1=logical([1 1 ; 1 1])
d=logical([ 1; 1 ])
c=stringMultBool(A,b1)
b2=logical([1 0 ; 1 0])
c2=stringMultBool(A,b2)

```

```

A =
    'a11'    'a12r'
    'a21'    'a22e'
B =
    'b11'    'b12r'
    'b21'    'b22e'
x =
    'a11'    'a12r'
    'a21'    'a22e'
y =
    'a11b11'    'a12rb12r'
    'a21b21'    'a22eb22e'
b1 =
     1     1
     1     1
d =
     1
     1
c =
    'a11a12r'    'a11a12r'
    'a21a22e'    'a21a22e'
b2 =
     1     0
     1     0
c2 =
    'a11a12r'    ''
    'a21a22e'    ''
>>

```

Appendix G

Engineering Process Model

G.1 Engineering Process Model Example - MATLAB Code

Contents

- Relation algebra MATLAB functionality
- Person executes task data and graph data output
- Compute transpose for task executed by person and graph data output
- Data set creates / modifies / reads data and graph output data
- Person - Data logical deductions
- Data - Tool(G) relations
- Union operations
- Transpose for tools used by tasks
- Union of data and person logic
- Person tool deduced logic
- Alternative data operated on by task
- Deduce links data - task via tool to determine data sequence
- Deduce links data - task via tool to determine data sequence

```
%.....  
% ProjecModelBuildingUpd.m  
% Project Model for typical Building Project  
%.....  
clc  
clear all  
format compact
```

Relation algebra MATLAB functionality

```
RelationalAlgebraBoolean
```

```
MATLAB implementation of relational algebra boolean matrix operations in inline functions
```

Person executes task data and graph data output

Define persons executes tasks PT Tasks: Conceptualise / Plan / Design / Specify & Document / Take off Quantities / Build & Construct

```
PexecutesT=([1 0 0 0 0 0 ; ... % Client Owner  
            1 1 0 0 0 0 ; ... % Architect  
            0 0 1 1 0 0 ; ... % Engineer  
            0 0 0 1 1 0 ; ... % Quantity Surveyor  
            0 0 0 0 0 1 ]) % Constructor/Contractor  
% TgfWrite(tgfFile,R,vertexLabels,edgeLabels)  
personLabels={'Client Owner','Architect','Engineer', ...  
             'Quantity Surveyor','Constructor/Contractor/Builder'}  
taskLabels={'Conceptualise','Plan','Eng Design', ...
```

```

        'Specify & Document', ...
        'Take off Quantities','Build & Construct'}
% add comment & heading strings at end of labels
vertexLabels=horzcat(personLabels,taskLabels',{'Person executes Task'})
isHomog=0
tgfWrite('YPexecutesT.tgf',PexecutesT,isHomog,vertexLabels,{})

```

```

PexecutesT =
    1     0     0     0     0     0
    1     1     0     0     0     0
    0     0     1     1     0     0
    0     0     0     1     1     0
    0     0     0     0     0     1
personLabels =
  Columns 1 through 4
    'Client Owner'    'Architect'    'Engineer'    'Quantity Surveyor'
  Column 5
    [1x30 char]
taskLabels =
  Columns 1 through 4
    'Conceptualise'    'Plan'    'Eng Design'    'Specify & Document'
  Columns 5 through 6
    'Take off Quantities'    'Build & Construct'
vertexLabels =
  Columns 1 through 4
    'Client Owner'    'Architect'    'Engineer'    'Quantity Surveyor'
  Columns 5 through 8
    [1x30 char]    'Conceptualise'    'Plan'    'Eng Design'
  Columns 9 through 11
    'Specify & Document'    'Take off Quantities'    'Build & Construct'
  Column 12
    [1x20 char]
isHomog =
    0
File YPexecutesT.tgf opened
nRows =
    5
nCols =
    6
File YPexecutesT.tgf closed

```

Compute transpose for task executed by person and graph data output

Compute tasks - persons TP

```

vertexLabels=horzcat(taskLabels,personLabels',{'Task executed by Person'})
TexecutedByP=transposeR(PexecutesT)
isHomog=0
tgfWrite('YTexecutedByP.tgf',TexecutedByP,isHomog,vertexLabels,{})

```

```

vertexLabels =
  Columns 1 through 4
    'Conceptualise'    'Plan'    'Eng Design'    'Specify & Document'
  Columns 5 through 7
    'Take off Quantities'    'Build & Construct'    'Client Owner'
  Columns 8 through 11
    'Architect'    'Engineer'    'Quantity Surveyor'    [1x30 char]
  Column 12
    [1x23 char]
TexecutedByP =
    1     1     0     0     0

```

```

0      1      0      0      0
0      0      1      0      0
0      0      1      1      0
0      0      0      1      0
0      0      0      0      1
isHomog =
0
File YTexecutedByP.tgf opened
nRows =
6
nCols =
5
File YTexecutedByP.tgf closed

```

Data set creates / modifies / reads data and graph output data

Task creates / modifies / reads data Concept plan = dwgs +document / Architects dwgs / Design calcs / Eng Plans / Specifications / Bill of Quantities / Construction pgms/sched / As built plans

```

dataLabels={'Concept plan','Architects dwgs','Design calculations', ...
            'Engineering dwgs','Specifications','Bill of Quantities', ...
            'Construction pgms/sched','As built dwgs'}
TcreatesD=([1 0 0 0 0 0 0 0 ; ... % Conceptualise
            0 1 0 0 0 0 0 0; ... % Plan
            0 0 1 1 0 0 0 0; ... % Engineering design
            0 0 0 0 1 0 0 0; ... % Specify and document
            0 0 0 0 0 1 0 0; ... % Take off quantities
            0 0 0 0 0 0 1 1]) % Construct

isHomog=0
vertexLabels=horzcat(taskLabels,dataLabels',{'Task creates Dataset'})
tgfWrite('YTcreatesD.tgf',TcreatesD,isHomog,vertexLabels,{})
% Compute transpose for data create - data created by task
DcreatedByT=transposeR(TcreatesD)
isHomog=0
vertexLabels=horzcat(dataLabels,taskLabels',{'Dataset created by Task'})
tgfWrite('YDcreatedByT.tgf',DcreatedByT,isHomog,vertexLabels,{})
TreadsD= ([0 0 0 0 0 0 0 0 ; ... % Conceptualise
            1 0 0 0 0 0 0 0; ... % Plan
            1 1 0 0 0 0 0 0; ... % Engineering design
            0 1 1 0 0 0 0 0; ... % Specify and document
            0 1 0 1 1 0 0 0; ... % Take off quantities
            0 1 0 1 1 1 1 0]) % Construct

isHomog=0
vertexLabels=horzcat(taskLabels,dataLabels',{'Task reads Dataset'})
tgfWrite('YTreadsD.tgf',TreadsD,isHomog,vertexLabels,{})
% Compute transpose for data read - data read by task
DreadByT=transposeR(TreadsD)
isHomog=0
vertexLabels=horzcat(dataLabels,taskLabels',{'Dataset read by Task'})
tgfWrite('YDreadByT.tgf',DreadByT,isHomog,vertexLabels,{})
% Concept plan / Architects plan / Design calcs / Eng Plans / Specifications
% / Bill of Quantities / As built plans
TmodifiesD= ([1 1 0 0 0 0 0 0; ... % Conceptualise
            0 1 0 0 0 0 0 0; ... % Plan
            0 0 1 1 0 0 0 0; ... % Engineering design
            0 0 0 0 1 0 0 0; ... % Specify and document
            0 0 0 0 0 1 0 0; ... % Take off quantities
            0 0 0 0 0 0 1 1]) % Construct

isHomog=0
vertexLabels=horzcat(taskLabels,dataLabels',{'Task modifies Dataset'})
tgfWrite('YTmodifiesD.tgf',TmodifiesD,isHomog,vertexLabels,{})
% Compute transpose for data modify - data modified by task
DmodifiedByT=transposeR(TmodifiesD)
isHomog=0
vertexLabels=horzcat(dataLabels,taskLabels',{'Dataset modified by Task'})
tgfWrite('YDmodifiedByT.tgf',DmodifiedByT,isHomog,vertexLabels,{})

```



```

dataLabels =
  Columns 1 through 3
    'Concept plan'      'Architects dwgs'      'Design calculations'
  Columns 4 through 6
    'Engineering dwgs'  'Specifications'    'Bill of Quantities'
  Columns 7 through 8
    [1x23 char]        'As built dwgs'
TcreatesD =
    1    0    0    0    0    0    0    0
    0    1    0    0    0    0    0    0
    0    0    1    1    0    0    0    0
    0    0    0    0    1    0    0    0
    0    0    0    0    0    1    0    0
    0    0    0    0    0    0    1    1
isHomog =
    0
vertexLabels =
  Columns 1 through 4
    'Conceptualise'    'Plan'      'Eng Design'    'Specify & Document'
  Columns 5 through 7
    'Take off Quantities'    'Build & Construct'    'Concept plan'
  Columns 8 through 10
    'Architects dwgs'    'Design calculations'    'Engineering dwgs'
  Columns 11 through 13
    'Specifications'    'Bill of Quantities'    [1x23 char]
  Columns 14 through 15
    'As built dwgs'    [1x20 char]
File YTcreatesD.tgf opened
nRows =
    6
nCols =
    8
File YTcreatesD.tgf closed
DcreatedByT =
    1    0    0    0    0    0
    0    1    0    0    0    0
    0    0    1    0    0    0
    0    0    1    0    0    0
    0    0    0    1    0    0
    0    0    0    0    1    0
    0    0    0    0    0    1
    0    0    0    0    0    1
isHomog =
    0
vertexLabels =
  Columns 1 through 3
    'Concept plan'      'Architects dwgs'      'Design calculations'
  Columns 4 through 6
    'Engineering dwgs'  'Specifications'    'Bill of Quantities'
  Columns 7 through 10
    [1x23 char]        'As built dwgs'    'Conceptualise'    'Plan'
  Columns 11 through 13
    'Eng Design'    'Specify & Document'    'Take off Quantities'
  Columns 14 through 15
    'Build & Construct'    [1x23 char]
File YDcreatedByT.tgf opened
nRows =
    8
nCols =
    6
File YDcreatedByT.tgf closed
TreadsD =
    0    0    0    0    0    0    0    0
    1    0    0    0    0    0    0    0
    1    1    0    0    0    0    0    0
    0    1    1    0    0    0    0    0
    0    1    0    1    1    0    0    0
    0    1    0    1    1    1    1    0
isHomog =
    0

```

```

vertexLabels =
  Columns 1 through 4
    'Conceptualise' 'Plan' 'Eng Design' 'Specify & Document'
  Columns 5 through 7
    'Take off Quantities' 'Build & Construct' 'Concept plan'
  Columns 8 through 10
    'Architects dwgs' 'Design calculations' 'Engineering dwgs'
  Columns 11 through 13
    'Specifications' 'Bill of Quantities' [1x23 char]
  Columns 14 through 15
    'As built dwgs' 'Task reads Dataset'
File YTreadsD.tgf opened
nRows =
    6
nCols =
    8
File YTreadsD.tgf closed
DreadByT =
    0     1     1     0     0     0
    0     0     1     1     1     1
    0     0     0     1     0     0
    0     0     0     0     1     1
    0     0     0     0     1     1
    0     0     0     0     0     1
    0     0     0     0     0     1
    0     0     0     0     0     0
isHomog =
    0
vertexLabels =
  Columns 1 through 3
    'Concept plan' 'Architects dwgs' 'Design calculations'
  Columns 4 through 6
    'Engineering dwgs' 'Specifications' 'Bill of Quantities'
  Columns 7 through 10
    [1x23 char] 'As built dwgs' 'Conceptualise' 'Plan'
  Columns 11 through 13
    'Eng Design' 'Specify & Document' 'Take off Quantities'
  Columns 14 through 15
    'Build & Construct' [1x20 char]
File YDreadByT.tgf opened
nRows =
    8
nCols =
    6
File YDreadByT.tgf closed
TmodifiesD =
    1     1     0     0     0     0     0     0
    0     1     0     0     0     0     0     0
    0     0     1     1     0     0     0     0
    0     0     0     0     1     0     0     0
    0     0     0     0     0     1     0     0
    0     0     0     0     0     0     1     1
isHomog =
    0
vertexLabels =
  Columns 1 through 4
    'Conceptualise' 'Plan' 'Eng Design' 'Specify & Document'
  Columns 5 through 7
    'Take off Quantities' 'Build & Construct' 'Concept plan'
  Columns 8 through 10
    'Architects dwgs' 'Design calculations' 'Engineering dwgs'
  Columns 11 through 13
    'Specifications' 'Bill of Quantities' [1x23 char]
  Columns 14 through 15
    'As built dwgs' [1x21 char]
File YTmodifiesD.tgf opened
nRows =
    6
nCols =
    8
File YTmodifiesD.tgf closed

```

```

DmodifiedByT =
    1     0     0     0     0     0
    1     1     0     0     0     0
    0     0     1     0     0     0
    0     0     1     0     0     0
    0     0     0     1     0     0
    0     0     0     0     1     0
    0     0     0     0     0     1
    0     0     0     0     0     1
isHomog =
    0
vertexLabels =
    Columns 1 through 3
        'Concept plan'      'Architects dwgs'      'Design calculations'
    Columns 4 through 6
        'Engineering dwgs'    'Specifications'    'Bill of Quantities'
    Columns 7 through 10
        [1x23 char]    'As built dwgs'    'Conceptualise'    'Plan'
    Columns 11 through 13
        'Eng Design'    'Specify & Document'    'Take off Quantities'
    Columns 14 through 15
        'Build & Construct'    [1x24 char]
File YDmodifiedByT.tgf opened
nRows =
    8
nCols =
    6
File YDmodifiedByT.tgf closed

```

Person - Data logical deductions

Person creates data

```

PcreatesD=productR(PexecutesT,TcreatesD)
isHomog=0
vertexLabels=horzcat(personLabels,dataLabels',{'Person creates Dataset'})
tgfWrite('YPcreatesD.tgf',PcreatesD,isHomog,vertexLabels,{})
% Data created by person
DcreatedByP=transposeR(PcreatesD)
isHomog=0
vertexLabels=horzcat(dataLabels,personLabels',{'Dataset created by Person'})
tgfWrite('YDcreatedByP.tgf',DcreatedByP,isHomog,vertexLabels,{})
% Person reading data
PreadsD=productR(PexecutesT,TreadsD)
isHomog=0
vertexLabels=horzcat(personLabels,dataLabels',{'Person reads Dataset'})
tgfWrite('YPreadsD.tgf',PreadsD,isHomog,vertexLabels,{})
% Data read by person
DreadByP=transposeR(PreadsD)
isHomog=0
vertexLabels=horzcat(dataLabels,personLabels',{'Dataset read by Person'})
tgfWrite('YDreadByP.tgf',DreadByP,isHomog,vertexLabels,{})
% Person modifies data
PmodifiesD=productR(PexecutesT,TmodifiesD)
isHomog=0
vertexLabels=horzcat(personLabels,dataLabels',{'Person modifies Dataset'})
tgfWrite('YDreadByP.tgf',PmodifiesD,isHomog,vertexLabels,{})
% Data modified by person
DmodifiedByP=transposeR(PmodifiesD)
isHomog=0
vertexLabels=horzcat(dataLabels,personLabels',{'Dataset modified by Person'})
tgfWrite('YDmodifiedByP.tgf',DmodifiedByP,isHomog,vertexLabels,{})

```

```

PcreatesD =
  1   0   0   0   0   0   0   0
  1   1   0   0   0   0   0   0
  0   0   1   1   1   0   0   0
  0   0   0   0   1   1   0   0
  0   0   0   0   0   0   1   1
isHomog =
  0
vertexLabels =
  Columns 1 through 4
    'Client Owner'    'Architect'    'Engineer'    'Quantity Surveyor'
  Columns 5 through 7
    [1x30 char]    'Concept plan'    'Architects dwgs'
  Columns 8 through 10
    'Design calculations'    'Engineering dwgs'    'Specifications'
  Columns 11 through 14
    'Bill of Quantities'    [1x23 char]    'As built dwgs'    [1x22 char]
File YPcreatesD.tgf opened
nRows =
  5
nCols =
  8
File YPcreatesD.tgf closed
DcreatedByP =
  1   1   0   0   0
  0   1   0   0   0
  0   0   1   0   0
  0   0   1   0   0
  0   0   1   1   0
  0   0   0   1   0
  0   0   0   0   1
  0   0   0   0   1
isHomog =
  0
vertexLabels =
  Columns 1 through 3
    'Concept plan'    'Architects dwgs'    'Design calculations'
  Columns 4 through 6
    'Engineering dwgs'    'Specifications'    'Bill of Quantities'
  Columns 7 through 10
    [1x23 char]    'As built dwgs'    'Client Owner'    'Architect'
  Columns 11 through 14
    'Engineer'    'Quantity Surveyor'    [1x30 char]    [1x25 char]
File YDcreatedByP.tgf opened
nRows =
  8
nCols =
  5
File YDcreatedByP.tgf closed
PreadsD =
  0   0   0   0   0   0   0   0
  1   0   0   0   0   0   0   0
  1   1   1   0   0   0   0   0
  0   1   1   1   1   0   0   0
  0   1   0   1   1   1   1   0
isHomog =
  0
vertexLabels =
  Columns 1 through 4
    'Client Owner'    'Architect'    'Engineer'    'Quantity Surveyor'
  Columns 5 through 7
    [1x30 char]    'Concept plan'    'Architects dwgs'
  Columns 8 through 10
    'Design calculations'    'Engineering dwgs'    'Specifications'
  Columns 11 through 14
    'Bill of Quantities'    [1x23 char]    'As built dwgs'    [1x20 char]
File YPreadsD.tgf opened
nRows =
  5
nCols =
  8
File YPreadsD.tgf closed
DreadByP =

```

```

0      1      1      0      0
0      0      1      1      1
0      0      1      1      0
0      0      0      1      1
0      0      0      1      1
0      0      0      0      1
0      0      0      0      1
0      0      0      0      0
isHomog =
0
vertexLabels =
Columns 1 through 3
'Concept plan'      'Architects dwgs'      'Design calculations'
Columns 4 through 6
'Engineering dwgs'      'Specifications'      'Bill of Quantities'
Columns 7 through 10
[1x23 char]      'As built dwgs'      'Client Owner'      'Architect'
Columns 11 through 14
'Engineer'      'Quantity Surveyor'      [1x30 char]      [1x22 char]
File YDreadByP.tgf opened
nRows =
8
nCols =
5
File YDreadByP.tgf closed
PmodifiesD =
1      1      0      0      0      0      0      0
1      1      0      0      0      0      0      0
0      0      1      1      1      0      0      0
0      0      0      0      1      1      0      0
0      0      0      0      0      0      1      1
isHomog =
0
vertexLabels =
Columns 1 through 4
'Client Owner'      'Architect'      'Engineer'      'Quantity Surveyor'
Columns 5 through 7
[1x30 char]      'Concept plan'      'Architects dwgs'
Columns 8 through 10
'Design calculations'      'Engineering dwgs'      'Specifications'
Columns 11 through 14
'Bill of Quantities'      [1x23 char]      'As built dwgs'      [1x23 char]
File YPmodifiesD.tgf opened
nRows =
5
nCols =
8
File YPmodifiesD.tgf closed
DmodifiedByP =
1      1      0      0      0
1      1      0      0      0
0      0      1      0      0
0      0      1      0      0
0      0      1      1      0
0      0      0      1      0
0      0      0      0      1
0      0      0      0      1
isHomog =
0
vertexLabels =
Columns 1 through 3
'Concept plan'      'Architects dwgs'      'Design calculations'
Columns 4 through 6
'Engineering dwgs'      'Specifications'      'Bill of Quantities'
Columns 7 through 10
[1x23 char]      'As built dwgs'      'Client Owner'      'Architect'
Columns 11 through 14
'Engineer'      'Quantity Surveyor'      [1x30 char]      [1x26 char]
File YDmodifiedByP.tgf opened
nRows =
8

```

```
nCols =
    5
File YDmodifiedByP.tgf closed
```

Data - Tool(G) relations

Tools: Text processor / CAD / Eng Design SW / Quantities SW / Construction Planning SW

```
toolLabels={'Text processor','CAD','Eng Design SW', ...
            'Quantities SW','Construction Planning SW'}
% Data requires tool
DrequiresG=([1 1 0 0 0 ; ... % Conceptual plan requirestxt processor & CAD
            0 1 0 0 0 ; ... % Arch plan rquires CAD processor
            0 0 1 0 0 ; ... % Design calcs require Eng Design SW
            0 1 0 0 0 ; ... % Engineering plans require CAD
            1 0 0 0 0 ; ... % Specifications require txt processor
            0 0 0 1 0 ; ... % Bill of Quantities requires off Quant software
            0 0 0 0 1 ; ... % Construction plans & schedules Constr SW
            0 1 0 0 0]) % As built plans require CAD
isHomog=0
vertexLabels=horzcat(dataLabels,toolLabels',{'Dataset requires Tool'})
tgfWrite('YDrequiresG.tgf',DrequiresG,isHomog,vertexLabels,{})
% Tool used on data - transpose Data requires tool
GoperatesOnD=transposeR(DrequiresG)
isHomog=0
vertexLabels=horzcat(toolLabels,dataLabels',{'Tool operates on Dataset'})
tgfWrite('YGoperatesOnD.tgf',GoperatesOnD,isHomog,vertexLabels,{})
```

```
toolLabels =
    Columns 1 through 4
    'Text processor'    'CAD'    'Eng Design SW'    'Quantities SW'
    Column 5
    [1x24 char]
DrequiresG =
    1    1    0    0    0
    0    1    0    0    0
    0    0    1    0    0
    0    1    0    0    0
    1    0    0    0    0
    0    0    0    1    0
    0    0    0    0    1
    0    1    0    0    0
isHomog =
    0
vertexLabels =
    Columns 1 through 3
    'Concept plan'    'Architects dwgs'    'Design calculations'
    Columns 4 through 6
    'Engineering dwgs'    'Specifications'    'Bill of Quantities'
    Columns 7 through 10
    [1x23 char]    'As built dwgs'    'Text processor'    'CAD'
    Columns 11 through 14
    'Eng Design SW'    'Quantities SW'    [1x24 char]    [1x21 char]
File YDrequiresG.tgf opened
nRows =
    8
nCols =
    5
File YDrequiresG.tgf closed
GoperatesOnD =
    1    0    0    0    1    0    0    0
    1    1    0    1    0    0    0    1
    0    0    1    0    0    0    0    0
    0    0    0    0    0    1    0    0
```

```

0      0      0      0      0      0      1      0
isHomog =
0
vertexLabels =
Columns 1 through 4
'Text processor'      'CAD'      'Eng Design SW'      'Quantities SW'
Columns 5 through 7
[1x24 char]      'Concept plan'      'Architects dwgs'
Columns 8 through 10
'Design calculations'      'Engineering dwgs'      'Specifications'
Columns 11 through 14
'Bill of Quantities'      [1x23 char]      'As built dwgs'      [1x24 char]
File YGoperatesOnD.tgf opened
nRows =
5
nCols =
8
File YGoperatesOnD.tgf closed

```

Union operations

Union of task - data (create & read & modify)

```

ToperatesOnD=unionR(unionR(TreadsD,TcreatesD),TmodifiesD)
isHomog=0
vertexLabels=horzcat(taskLabels,dataLabels',{'Task operates on Dataset'})
tgfWrite('YToperatesOnD.tgf',ToperatesOnD,isHomog,vertexLabels,{})
% Union of data (create & read & modify)- task
DoperatedOnByT=unionR(unionR(DreadByT,DcreatedByT),DmodifiedByT)
isHomog=0
vertexLabels=horzcat(dataLabels,taskLabels',{'Data operated on by Task'})
tgfWrite('YDoperatedOnByT.tgf',DoperatedOnByT,isHomog,vertexLabels,{})
% Task uses / requires tool deduced from Task - dataset & Dataset-Tool union
TusesG=productR(ToperatesOnD,DrequiresG)
isHomog=0
vertexLabels=horzcat(taskLabels,toolLabels',{'Task uses/requires Tool'})
tgfWrite('YTusesG.tgf',TusesG,isHomog,vertexLabels,{})

```

```

ToperatesOnD =
1      1      0      0      0      0      0      0
1      1      0      0      0      0      0      0
1      1      1      1      0      0      0      0
0      1      1      0      1      0      0      0
0      1      0      1      1      1      0      0
0      1      0      1      1      1      1      1
isHomog =
0
vertexLabels =
Columns 1 through 4
'Conceptualise'      'Plan'      'Eng Design'      'Specify & Document'
Columns 5 through 7
'Take off Quantities'      'Build & Construct'      'Concept plan'
Columns 8 through 10
'Architects dwgs'      'Design calculations'      'Engineering dwgs'
Columns 11 through 13
'Specifications'      'Bill of Quantities'      [1x23 char]
Columns 14 through 15
'As built dwgs'      [1x24 char]
File YToperatesOnD.tgf opened
nRows =
6
nCols =
8
File YToperatesOnD.tgf closed

```

```

DoperatedOnByT =
  1   1   1   0   0   0
  1   1   1   1   1   1
  0   0   1   1   0   0
  0   0   1   0   1   1
  0   0   0   1   1   1
  0   0   0   0   1   1
  0   0   0   0   0   1
  0   0   0   0   0   1

isHomog =
  0
vertexLabels =
  Columns 1 through 3
    'Concept plan'    'Architects dwgs'    'Design calculations'
  Columns 4 through 6
    'Engineering dwgs'    'Specifications'    'Bill of Quantities'
  Columns 7 through 10
    [1x23 char]    'As built dwgs'    'Conceptualise'    'Plan'
  Columns 11 through 13
    'Eng Design'    'Specify & Document'    'Take off Quantities'
  Columns 14 through 15
    'Build & Construct'    [1x24 char]
File YDoperatedOnByT.tgf opened
nRows =
  8
nCols =
  6
File YDoperatedOnByT.tgf closed
TusesG =
  1   1   0   0   0
  1   1   0   0   0
  1   1   1   0   0
  1   1   1   0   0
  1   1   0   1   0
  1   1   0   1   1

isHomog =
  0
vertexLabels =
  Columns 1 through 4
    'Conceptualise'    'Plan'    'Eng Design'    'Specify & Document'
  Columns 5 through 8
    'Take off Quantities'    'Build & Construct'    'Text processor'    'CAD'
  Columns 9 through 12
    'Eng Design SW'    'Quantities SW'    [1x24 char]    [1x23 char]
File YTusesG.tgf opened
nRows =
  6
nCols =
  5
File YTusesG.tgf closed

```

Transpose for tools used by tasks

```

GusedByT=transposeR(TusesG)
isHomog=0
vertexLabels=horzcat(toolLabels,taskLabels',{'Tool used by Task'})
tgfWrite('YGusedByT.tgf',GusedByT,isHomog,vertexLabels,{'})

```

```

GusedByT =
  1   1   1   1   1   1
  1   1   1   1   1   1
  0   0   1   1   0   0
  0   0   0   0   1   1
  0   0   0   0   0   1

isHomog =

```



```

0
vertexLabels =
Columns 1 through 4
    'Text processor'      'CAD'      'Eng Design SW'      'Quantities SW'
Columns 5 through 8
    [1x24 char]      'Conceptualise'      'Plan'      'Eng Design'
Columns 9 through 11
    'Specify & Document'      'Take off Quantities'      'Build & Construct'
Column 12
    'Tool used by Task'
File YGusedByT.tgf opened
nRows =
    5
nCols =
    6
File YGusedByT.tgf closed

```

Union of data and person logic

Union of data (create & read & modify)- person

```

PoperatesOnD=unionR(unionR(PreadsD,PcreatesD),PmodifiesD)
isHomog=0
vertexLabels=horzcat(personLabels,dataLabels',{'Person operates on Dataset'})
tgfWrite('YPoperatesOnD.tgf',PoperatesOnD,isHomog,vertexLabels,{})
% Union of data (create & read & modify) - person - Data operated on by
% person
DoperatedOnByP=unionR(unionR(DreadByP,DcreatedByP),DmodifiedByP)
isHomog=0
vertexLabels=horzcat(dataLabels,personLabels',{'Data operated on by Person'})
tgfWrite('YDoperatedOnByP.tgf',DoperatedOnByP,isHomog,vertexLabels,{})

```

```

PoperatesOnD =
    1     1     0     0     0     0     0     0
    1     1     0     0     0     0     0     0
    1     1     1     1     1     0     0     0
    0     1     1     1     1     1     0     0
    0     1     0     1     1     1     1     1
isHomog =
    0
vertexLabels =
Columns 1 through 4
    'Client Owner'      'Architect'      'Engineer'      'Quantity Surveyor'
Columns 5 through 7
    [1x30 char]      'Concept plan'      'Architects dwgs'
Columns 8 through 10
    'Design calculations'      'Engineering dwgs'      'Specifications'
Columns 11 through 14
    'Bill of Quantities'      [1x23 char]      'As built dwgs'      [1x26 char]
File YPoperatesOnD.tgf opened
nRows =
    5
nCols =
    8
File YPoperatesOnD.tgf closed
DoperatedOnByP =
    1     1     1     0     0
    1     1     1     1     1
    0     0     1     1     0
    0     0     1     1     1
    0     0     0     1     1
    0     0     0     0     1
    0     0     0     0     1

```

```

isHomog =
    0
vertexLabels =
    Columns 1 through 3
        'Concept plan'      'Architects dwgs'      'Design calculations'
    Columns 4 through 6
        'Engineering dwgs'  'Specifications'  'Bill of Quantities'
    Columns 7 through 10
        [1x23 char]        'As built dwgs'    'Client Owner'      'Architect'
    Columns 11 through 14
        'Engineer'         'Quantity Surveyor' [1x30 char]        [1x26 char]
File YDoperatedOnByP.tgf opened
nRows =
    8
nCols =
    5
File YDoperatedOnByP.tgf closed

```

Person tool deduced logic

Person uses / requires tool deduced from Person - dataset & Dataset-Tool union

```

PusesG=productR(PoperatesOnD,DrequiresG)
isHomog=0
vertexLabels=horzcat(personLabels,toolLabels',{'Person uses/requires Tool'})
tgfWrite('YPusesG.tgf',PusesG,isHomog,vertexLabels,{})
% Tools used by persons
GusedByP=transposeR(PusesG)
isHomog=0
vertexLabels=horzcat(toolLabels,personLabels',{'Tool used by Person'})
tgfWrite('YGusedByP.tgf',GusedByP,isHomog,vertexLabels,{})

```

```

PusesG =
     1     1     0     0     0
     1     1     0     0     0
     1     1     1     0     0
     1     1     1     1     0
     1     1     0     1     1
isHomog =
    0
vertexLabels =
    Columns 1 through 4
        'Client Owner'      'Architect'      'Engineer'      'Quantity Surveyor'
    Columns 5 through 8
        [1x30 char]        'Text processor'  'CAD'          'Eng Design SW'
    Columns 9 through 11
        'Quantities SW'    [1x24 char]      [1x25 char]
File YPusesG.tgf opened
nRows =
    5
nCols =
    5
File YPusesG.tgf closed
GusedByP =
     1     1     1     1     1
     1     1     1     1     1
     0     0     1     1     0
     0     0     0     1     1
     0     0     0     0     1
isHomog =
    0
vertexLabels =
    Columns 1 through 4
        'Text processor'      'CAD'          'Eng Design SW'      'Quantities SW'

```

```

Columns 5 through 8
[1x24 char]      'Client Owner'      'Architect'      'Engineer'
Columns 9 through 11
'Quantity Surveyor' [1x30 char]      'Tool used by Person'
File YGusedByP.tgf opened
nRows =
    5
nCols =
    5
File YGusedByP.tgf closed

```

Alternative data operated on by task

Alternative to data operated on by task - transpose

```

DoperatedOnByPAlt=transposeR(PoperatesOnD)
isHomog=0
vertexLabels=horzcat(dataLabels,personLabels',{'Data operated on by Person (T)'})
tgfWrite('YDoperatedOnByPAlt.tgf',DoperatedOnByPAlt,isHomog,vertexLabels,{})

```

```

DoperatedOnByPAlt =
    1    1    1    0    0
    1    1    1    1    1
    0    0    1    1    0
    0    0    1    1    1
    0    0    1    1    1
    0    0    0    1    1
    0    0    0    0    1
    0    0    0    0    1
isHomog =
    0
vertexLabels =
Columns 1 through 3
'Concept plan'      'Architects dwgs'      'Design calculations'
Columns 4 through 6
'Engineering dwgs'      'Specifications'      'Bill of Quantities'
Columns 7 through 10
[1x23 char]      'As built dwgs'      'Client Owner'      'Architect'
Columns 11 through 14
'Engineer'      'Quantity Surveyor'      [1x30 char]      [1x30 char]
File YDoperatedOnByPAlt.tgf opened
nRows =
    8
nCols =
    5
File YDoperatedOnByPAlt.tgf closed

```

Deduce links data - task via tool to determine data sequence

Dataset - Task link deduced via tool

```

DoperatedOnByTViaTool=productR(DrequiresG,GusedByT)
isHomog=0
vertexLabels=horzcat(dataLabels,taskLabels',{'Data operated on by Task via Tool'})
tgfWrite('YDoperatedOnByTViaTool.tgf',DoperatedOnByTViaTool,isHomog,vertexLabels,{})
%
% Dataset sequence history - via person - read only
DhistoryDReadPerson=productR(DcreatedByP,PreadsD)
% Transpose
DhistoryDReadPerson=transposeR(DhistoryDReadPerson)

```

```

isHomog=1
vertexLabels=horzcat(dataLabels, ...
{'Dataset history with Dataset via Persons - Read only'})
tgfWrite('YDhistoryDviaReadPerson.tgf',DhistoryDReadPerson,isHomog,vertexLabels,{})

```

```

DoperatedOnByTViaTool =
    1     1     1     1     1     1
    1     1     1     1     1     1
    0     0     1     1     0     0
    1     1     1     1     1     1
    1     1     1     1     1     1
    0     0     0     0     1     1
    0     0     0     0     0     1
    1     1     1     1     1     1
isHomog =
    0
vertexLabels =
    Columns 1 through 3
    'Concept plan'      'Architects dwgs'      'Design calculations'
    Columns 4 through 6
    'Engineering dwgs'   'Specifications'      'Bill of Quantities'
    Columns 7 through 10
    [1x23 char]          'As built dwgs'      'Conceptualise'      'Plan'
    Columns 11 through 13
    'Eng Design'         'Specify & Document'  'Take off Quantities'
    Columns 14 through 15
    'Build & Construct'   [1x33 char]
File YDoperatedOnByTViaTool.tgf opened
nRows =
    8
nCols =
    6
File YDoperatedOnByTViaTool.tgf closed
DhistoryDReadPerson =
    1     0     0     0     0     0     0     0
    1     0     0     0     0     0     0     0
    1     1     1     0     0     0     0     0
    1     1     1     0     0     0     0     0
    1     1     1     1     1     0     0     0
    0     1     1     1     1     0     0     0
    0     1     0     1     1     1     1     0
    0     1     0     1     1     1     1     0
DhistoryDReadPerson =
    1     1     1     1     1     0     0     0
    0     0     1     1     1     1     1     1
    0     0     1     1     1     1     0     0
    0     0     0     0     1     1     1     1
    0     0     0     0     1     1     1     1
    0     0     0     0     0     0     1     1
    0     0     0     0     0     0     1     1
    0     0     0     0     0     0     0     0
isHomog =
    1
vertexLabels =
    Columns 1 through 3
    'Concept plan'      'Architects dwgs'      'Design calculations'
    Columns 4 through 6
    'Engineering dwgs'   'Specifications'      'Bill of Quantities'
    Columns 7 through 9
    [1x23 char]          'As built dwgs'      [1x52 char]
File YDhistoryDviaReadPerson.tgf opened
nRows =
    8
nCols =
    8
File YDhistoryDviaReadPerson.tgf closed

```

Deduce links data - task via tool to determine data sequence

Dataset sequence history - via person - read & modify

```

PreadsUnionmodifiesD=transposeR(unionR(DreadByP,DmodifiedByP))
% PreadsUnionmodifiesD=transposeR(DreadByP)
isHomog=0
vertexLabels=horzcat(personLabels,dataLabels',{'Person reads/modifies data union'})
tgfWrite('YPreadsUnionModifiesD.tgf',PreadsUnionmodifiesD,isHomog,vertexLabels,{})
DhistoryDPerson=productR(DcreatedByP,PreadsUnionmodifiesD)
% Transpose
DhistoryDPerson=transposeR(DhistoryDPerson)
isHomog=1
vertexLabels=horzcat(dataLabels',{'Dataset history with Dataset via Persons'})
tgfWrite('YDhistoryDviaPerson.tgf',DhistoryDPerson,isHomog,vertexLabels,{})
%
% Dataset sequence history - via task - read only
DhistoryDReadTask=productR(DcreatedByT,TreadsD)
% Transpose
DhistoryDReadTask=transposeR(DhistoryDReadTask)
isHomog=1
vertexLabels=horzcat(dataLabels, ...
{'Dataset history with Dataset via Tasks - Read only'})
tgfWrite('YDhistoryDviaReadTask.tgf',DhistoryDReadTask,isHomog,vertexLabels,{})
% Dataset sequence history - via task - read & modify
TreadsUnionmodifiesD=transposeR(unionR(DreadByT,DmodifiedByT))
isHomog=0
vertexLabels=horzcat(taskLabels,dataLabels',{'Task reads/modifies data union'})
tgfWrite('YTreadsUnionModifiesD.tgf',TreadsUnionmodifiesD,isHomog,vertexLabels,{})
DhistoryDTask=productR(DcreatedByT,TreadsUnionmodifiesD)
isHomog=1
vertexLabels=horzcat(dataLabels',{'Dataset history with Dataset via Task'})
tgfWrite('YDhistoryDviaTask.tgf',DhistoryDTask,isHomog,vertexLabels,{})
% Person loading
DreadByUnionmodifiedByP=unionR(DreadByP,DmodifiedByP)
isHomog=0
vertexLabels=horzcat(dataLabels,personLabels',{'Data read/modify by Persons union'})
tgfWrite('YDreadbyUnionModifyByP.tgf',DreadByUnionmodifiedByP,isHomog,vertexLabels,{})
PloadingP=productR(PcreatesD,unionR(DreadByP,DmodifiedByP))
isHomog=1
vertexLabels=horzcat(personLabels',{'Persons loading with Persons'})
tgfWrite('YPloadingP.tgf',PloadingP,isHomog,vertexLabels,{})
% Tool loading - read & modify implied by Dataset requires Tool
DreadByG=DrequiresG
DmodifiedByG=DrequiresG
GcreatesD=productR(GusedByP,PcreatesD)
GcreatesD=transposeR(DrequiresG)
DreadByUnionmodifiedByG=unionR(DreadByG,DmodifiedByG)
isHomog=0
vertexLabels=horzcat(dataLabels,toolLabels',{'Data read/modify by Tools union'})
tgfWrite('YDreadbyUnionModifyByG.tgf',DreadByUnionmodifiedByG,isHomog,vertexLabels,{})
GloadingG=productR(GcreatesD,unionR(DreadByG,DmodifiedByG))
isHomog=1
vertexLabels=horzcat(toolLabels',{'Tool loading with Tool'})
tgfWrite('YGloadingG.tgf',GloadingG,isHomog,vertexLabels,{})
%.....
% Empty relation
% Test person intersections - resource loading
PavailableWithP=intersectionR(intersectionR(intersectionR(intersectionR( ...
    productR(PusesG,GusedByP), ...
    productR(PexecutesT,TexecutedByP)), ...
    productR(PcreatesD,DcreatedByP)), ...
    productR(PmodifiesD,DmodifiedByP)), ...
    productR(PreadsD,DreadByP))
isHomog=1
vertexLabels=horzcat(personLabels',{'Persons available with Persons'})
tgfWrite('YPavailableWithP.tgf',PavailableWithP,isHomog,vertexLabels,{})
% Test person union - resource loading
PtestunionP=unionR(unionR(unionR(unionR( ...
    productR(PusesG,GusedByP), ...
    productR(PexecutesT,TexecutedByP)), ...

```

```

        productR(PCreatesD,DcreatedByP)), ...
        productR(PmodifiesD,DmodifiedByP)), ...
        productR(PreadsD,DreadByP))
isHomog=1
vertexLabels=horzcat(personLabels',{'Persons test union with Persons'})
tgfWrite('YPtestunionP.tgf',PtestunionP,isHomog,vertexLabels,{})
% Test task intersections - task sequence
TavailableWithT=intersectionR(intersectionR(intersectionR(intersectionR( ...
        productR(TusesG,GusedByT), ...
        productR(TexecutedByP,PexecutesT)), ...
        productR(TcreatesD,DcreatedByT)), ...
        productR(TmodifiesD,DmodifiedByT)), ...
        productR(TreadsD,DreadByT))
isHomog=1
vertexLabels=horzcat(taskLabels',{'Tasks available with Tasks'})
tgfWrite('YTavailableWithT.tgf',TavailableWithT,isHomog,vertexLabels,{})
% Test task union - task sequence
TtestunionT=unionR(unionR(unionR(unionR( ...
        productR(TusesG,GusedByT), ...
        productR(TexecutedByP,PexecutesT)), ...
        productR(TcreatesD,DcreatedByT)), ...
        productR(TmodifiesD,DmodifiedByT)), ...
        productR(TreadsD,DreadByT))
DreadByUnionmodifiedByT=unionR(DreadByT,DmodifiedByT)
isHomog=0
vertexLabels=horzcat(dataLabels,taskLabels',{'Data read/modify by Task union'})
tgfWrite('YDreadbyUnionModifyByT.tgf',DreadByUnionmodifiedByT,isHomog,vertexLabels,{})
isHomog=1
vertexLabels=horzcat(taskLabels',{'Tasks test union with Tasks'})
tgfWrite('YTtestunionT.tgf',TtestunionT,isHomog,vertexLabels,{})
TsequenceT=productR(TcreatesD,unionR(DreadByT,DmodifiedByT))
isHomog=1
vertexLabels=horzcat(taskLabels',{'Tasks sequence with Tasks'})
tgfWrite('YTsequenceT.tgf',TsequenceT,isHomog,vertexLabels,{})
% Product to level 10
T=TsequenceT
% T=difference(T,identity(size(T)))
[Tn,Tplus]=RelationProduct(TsequenceT,10)
Tcycl=intersectionR(Tplus,transpose(Tplus))
TR=unionR(T,identityB(size(T)))
TS=unionR(T,transposeR(T))
[TSn,TSplus]=Relationproduct(TS,10)
% Only upper triangular applies
TX=differenceR(T,identityB(size(T)))
[Taskschedule]=TopolSortBFS(triu(TX))
% Other tested relations
% Empty relation
% Test person intersections - resource loading
PavailableWithP=intersectionR(intersectionR(intersectionR(intersectionR( ...
        productR(PusesG,GusedByP), ...
        productR(PexecutesT,TexecutedByP)), ...
        productR(PCreatesD,DcreatedByP)), ...
        productR(PmodifiesD,DmodifiedByP)), ...
        productR(PreadsD,DreadByP))
isHomog=1
vertexLabels=horzcat(personLabels',{'Persons available with Persons'})
tgfWrite('YPavailableWithP.tgf',PavailableWithP,isHomog,vertexLabels,{})
% Test person union - resource loading
PtestunionP=unionR(unionR(unionR(unionR( ...
        productR(PusesG,GusedByP), ...
        productR(PexecutesT,TexecutedByP)), ...
        productR(PCreatesD,DcreatedByP)), ...
        productR(PmodifiesD,DmodifiedByP)), ...
        productR(PreadsD,DreadByP))
isHomog=1
vertexLabels=horzcat(personLabels',{'Persons test union with Persons'})
tgfWrite('YPtestunionP.tgf',PtestunionP,isHomog,vertexLabels,{})
% Test task intersections - task sequence
TavailableWithT=intersectionR(intersectionR(intersectionR(intersectionR( ...
        productR(TusesG,GusedByT), ...
        productR(TexecutedByP,PexecutesT)), ...

```

```

        productR(TcreatesD,DcreatedByT)), ...
        productR(TmodifiesD,DmodifiedByT)), ...
        productR(TreadsD,DreadByT))
isHomog=1
vertexLabels=horzcat(taskLabels',{'Tasks available with Tasks'})
tgfWrite('YTavailableWithT.tgf',TavailableWithT,isHomog,vertexLabels,{})
% Test task union - task sequence
TtestunionT=unionR(unionR(unionR(unionR( ...
        productR(TusesG,GusedByT), ...
        productR(TexecutedByP,PexecutesT)), ...
        productR(TcreatesD,DcreatedByT)), ...
        productR(TmodifiesD,DmodifiedByT)), ...
        productR(TreadsD,DreadByT))

```

```

PreadsUnionmodifiesD =
    1    1    0    0    0    0    0    0
    1    1    0    0    0    0    0    0
    1    1    1    1    1    0    0    0
    0    1    1    1    1    1    0    0
    0    1    0    1    1    1    1    1
isHomog =
    0
vertexLabels =
    Columns 1 through 4
        'Client Owner'    'Architect'    'Engineer'    'Quantity Surveyor'
    Columns 5 through 7
    [1x30 char]    'Concept plan'    'Architects dwgs'
    Columns 8 through 10
        'Design calculations'    'Engineering dwgs'    'Specifications'
    Columns 11 through 14
        'Bill of Quantities'    [1x23 char]    'As built dwgs'    [1x32 char]
File YPreadsUnionModifiesD.tgf opened
nRows =
    5
nCols =
    8
File YPreadsUnionModifiesD.tgf closed
DhistoryDPerson =
    1    1    0    0    0    0    0    0
    1    1    0    0    0    0    0    0
    1    1    1    1    1    0    0    0
    1    1    1    1    1    0    0    0
    1    1    1    1    1    1    0    0
    0    1    1    1    1    1    0    0
    0    1    0    1    1    1    1    1
    0    1    0    1    1    1    1    1
DhistoryDPerson =
    1    1    1    1    1    0    0    0
    1    1    1    1    1    1    1    1
    0    0    1    1    1    1    0    0
    0    0    1    1    1    1    1    1
    0    0    1    1    1    1    1    1
    0    0    0    0    1    1    1    1
    0    0    0    0    0    0    1    1
    0    0    0    0    0    0    1    1
isHomog =
    1
vertexLabels =
    Columns 1 through 3
        'Concept plan'    'Architects dwgs'    'Design calculations'
    Columns 4 through 6
        'Engineering dwgs'    'Specifications'    'Bill of Quantities'
    Columns 7 through 9
    [1x23 char]    'As built dwgs'    [1x40 char]
File YDhistoryDviaPerson.tgf opened
nRows =
    8
nCols =

```

```

8
File YDhistoryDviaPerson.tgf closed
DhistoryDReadTask =
  0  0  0  0  0  0  0  0
  1  0  0  0  0  0  0  0
  1  1  0  0  0  0  0  0
  1  1  0  0  0  0  0  0
  0  1  1  0  0  0  0  0
  0  1  0  1  1  0  0  0
  0  1  0  1  1  1  1  0
  0  1  0  1  1  1  1  0
DhistoryDReadTask =
  0  1  1  1  0  0  0  0
  0  0  1  1  1  1  1  1
  0  0  0  0  1  0  0  0
  0  0  0  0  0  1  1  1
  0  0  0  0  0  1  1  1
  0  0  0  0  0  0  1  1
  0  0  0  0  0  0  1  1
  0  0  0  0  0  0  0  0
isHomog =
  1
vertexLabels =
  Columns 1 through 3
    'Concept plan'    'Architects dwgs'    'Design calculations'
  Columns 4 through 6
    'Engineering dwgs'    'Specifications'    'Bill of Quantities'
  Columns 7 through 9
    [1x23 char]    'As built dwgs'    [1x50 char]
File YDhistoryDviaReadTask.tgf opened
nRows =
  8
nCols =
  8
File YDhistoryDviaReadTask.tgf closed
TreadsUnionmodifiesD =
  1  1  0  0  0  0  0  0
  1  1  0  0  0  0  0  0
  1  1  1  1  0  0  0  0
  0  1  1  0  1  0  0  0
  0  1  0  1  1  1  0  0
  0  1  0  1  1  1  1  1
isHomog =
  0
vertexLabels =
  Columns 1 through 4
    'Conceptualise'    'Plan'    'Eng Design'    'Specify & Document'
  Columns 5 through 7
    'Take off Quantities'    'Build & Construct'    'Concept plan'
  Columns 8 through 10
    'Architects dwgs'    'Design calculations'    'Engineering dwgs'
  Columns 11 through 13
    'Specifications'    'Bill of Quantities'    [1x23 char]
  Columns 14 through 15
    'As built dwgs'    [1x30 char]
File YTreadsUnionModifiesD.tgf opened
nRows =
  6
nCols =
  8
File YTreadsUnionModifiesD.tgf closed
DhistoryDTask =
  1  1  0  0  0  0  0  0
  1  1  0  0  0  0  0  0
  1  1  1  1  0  0  0  0
  1  1  1  1  0  0  0  0
  0  1  1  0  1  0  0  0
  0  1  0  1  1  1  0  0
  0  1  0  1  1  1  1  1
  0  1  0  1  1  1  1  1
isHomog =

```



```

1
vertexLabels =
Columns 1 through 3
'Concept plan'      'Architects dwgs'      'Design calculations'
Columns 4 through 6
'Engineering dwgs'  'Specifications'      'Bill of Quantities'
Columns 7 through 9
[1x23 char]      'As built dwgs'      [1x37 char]
File YDhistoryDviaTask.tgf opened
nRows =
8
nCols =
8
File YDhistoryDviaTask.tgf closed
DreadByUnionmodifiedByP =
1      1      1      0      0
1      1      1      1      1
0      0      1      1      0
0      0      1      1      1
0      0      1      1      1
0      0      0      1      1
0      0      0      0      1
0      0      0      0      1
isHomog =
0
vertexLabels =
Columns 1 through 3
'Concept plan'      'Architects dwgs'      'Design calculations'
Columns 4 through 6
'Engineering dwgs'  'Specifications'      'Bill of Quantities'
Columns 7 through 10
[1x23 char]      'As built dwgs'      'Client Owner'      'Architect'
Columns 11 through 14
'Engineer'      'Quantity Surveyor'      [1x30 char]      [1x33 char]
File YDreadbyUnionModifyByP.tgf opened
nRows =
8
nCols =
5
File YDreadbyUnionModifyByP.tgf closed
PloadingP =
1      1      1      0      0
1      1      1      1      1
0      0      1      1      1
0      0      1      1      1
0      0      0      0      1
isHomog =
1
vertexLabels =
Columns 1 through 4
'Client Owner'      'Architect'      'Engineer'      'Quantity Surveyor'
Columns 5 through 6
[1x30 char]      [1x28 char]
File YPloadingP.tgf opened
nRows =
5
nCols =
5
File YPloadingP.tgf closed
DreadByG =
1      1      0      0      0
0      1      0      0      0
0      0      1      0      0
0      1      0      0      0
1      0      0      0      0
0      0      0      1      0
0      0      0      0      1
0      1      0      0      0
DmodifiedByG =
1      1      0      0      0
0      1      0      0      0

```

```

0      0      1      0      0
0      1      0      0      0
1      0      0      0      0
0      0      0      1      0
0      0      0      0      1
0      1      0      0      0
GcreatesD =
1      1      1      1      1      1      1      1
1      1      1      1      1      1      1      1
0      0      1      1      1      1      0      0
0      0      0      0      1      1      1      1
0      0      0      0      0      0      1      1
GcreatesD =
1      0      0      0      1      0      0      0
1      1      0      1      0      0      0      1
0      0      1      0      0      0      0      0
0      0      0      0      0      1      0      0
0      0      0      0      0      0      1      0
DreadByUnionmodifiedByG =
1      1      0      0      0
0      1      0      0      0
0      0      1      0      0
0      1      0      0      0
1      0      0      0      0
0      0      0      1      0
0      0      0      0      1
0      1      0      0      0
isHomog =
0
vertexLabels =
Columns 1 through 3
'Concept plan'      'Architects dwgs'      'Design calculations'
Columns 4 through 6
'Engineering dwgs'      'Specifications'      'Bill of Quantities'
Columns 7 through 10
[1x23 char]      'As built dwgs'      'Text processor'      'CAD'
Columns 11 through 14
'Eng Design SW'      'Quantities SW'      [1x24 char]      [1x31 char]
File YDreadbyUnionModifyByG.tgf opened
nRows =
8
nCols =
5
File YDreadbyUnionModifyByG.tgf closed
GloadingG =
1      1      0      0      0
1      1      0      0      0
0      0      1      0      0
0      0      0      1      0
0      0      0      0      1
isHomog =
1
vertexLabels =
Columns 1 through 4
'Text processor'      'CAD'      'Eng Design SW'      'Quantities SW'
Columns 5 through 6
[1x24 char]      [1x22 char]
File YGloadingG.tgf opened
nRows =
5
nCols =
5
File YGloadingG.tgf closed
PavailableWithP =
0      0      0      0      0
0      1      0      0      0
0      0      1      1      0
0      0      1      1      0
0      0      0      0      1
isHomog =
1

```

```

vertexLabels =
  Columns 1 through 4
    'Client Owner'    'Architect'    'Engineer'    'Quantity Surveyor'
  Columns 5 through 6
    [1x30 char]    [1x30 char]
File YPavailableWithP.tgf opened
nRows =
    5
nCols =
    5
File YPavailableWithP.tgf closed
PtestunionP =
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
isHomog =
    1
vertexLabels =
  Columns 1 through 4
    'Client Owner'    'Architect'    'Engineer'    'Quantity Surveyor'
  Columns 5 through 6
    [1x30 char]    [1x31 char]
File YPtestunionP.tgf opened
nRows =
    5
nCols =
    5
File YPtestunionP.tgf closed
TavailableWithT =
    0    0    0    0    0    0
    0    1    0    0    0    0
    0    0    1    0    0    0
    0    0    0    1    0    0
    0    0    0    0    1    0
    0    0    0    0    0    1
isHomog =
    1
vertexLabels =
  Columns 1 through 4
    'Conceptualise'    'Plan'    'Eng Design'    'Specify & Document'
  Columns 5 through 7
    'Take off Quantities'    'Build & Construct'    [1x26 char]
File YTavailableWithT.tgf opened
nRows =
    6
nCols =
    6
File YTavailableWithT.tgf closed
TtestunionT =
    1    1    1    1    1    1
    1    1    1    1    1    1
    1    1    1    1    1    1
    1    1    1    1    1    1
    1    1    1    1    1    1
    1    1    1    1    1    1
DreadByUnionmodifiedByT =
    1    1    1    0    0    0
    1    1    1    1    1    1
    0    0    1    1    0    0
    0    0    1    0    1    1
    0    0    0    1    1    1
    0    0    0    0    1    1
    0    0    0    0    0    1
    0    0    0    0    0    1
isHomog =
    0
vertexLabels =
  Columns 1 through 3
    'Concept plan'    'Architects dwgs'    'Design calculations'

```

```

Columns 4 through 6
'Engineering dwgs'      'Specifications'      'Bill of Quantities'
Columns 7 through 10
[1x23 char]      'As built dwgs'      'Conceptualise'      'Plan'
Columns 11 through 13
'Eng Design'      'Specify & Document'      'Take off Quantities'
Columns 14 through 15
'Build & Construct'      [1x30 char]
File YDreadbyUnionModifyByT.tgf opened
nRows =
8
nCols =
6
File YDreadbyUnionModifyByT.tgf closed
isHomog =
1
vertexLabels =
Columns 1 through 4
'Conceptualise'      'Plan'      'Eng Design'      'Specify & Document'
Columns 5 through 7
'Take off Quantities'      'Build & Construct'      [1x27 char]
File YTtestunionT.tgf opened
nRows =
6
nCols =
6
File YTtestunionT.tgf closed
TsequenceT =
1      1      1      0      0      0
1      1      1      1      1      1
0      0      1      1      1      1
0      0      0      1      1      1
0      0      0      0      1      1
0      0      0      0      0      1
isHomog =
1
vertexLabels =
Columns 1 through 4
'Conceptualise'      'Plan'      'Eng Design'      'Specify & Document'
Columns 5 through 7
'Take off Quantities'      'Build & Construct'      [1x25 char]
File YTsequenceT.tgf opened
nRows =
6
nCols =
6
File YTsequenceT.tgf closed
T =
1      1      1      0      0      0
1      1      1      1      1      1
0      0      1      1      1      1
0      0      0      1      1      1
0      0      0      0      1      1
0      0      0      0      0      1
MATLAB implementation of relational algebra boolean matrix operations in inline functions
Tn =
1      1      1      1      1      1
1      1      1      1      1      1
0      0      1      1      1      1
0      0      0      1      1      1
0      0      0      0      1      1
0      0      0      0      0      1
Tplus =
1      1      1      1      1      1
1      1      1      1      1      1
0      0      1      1      1      1
0      0      0      1      1      1
0      0      0      0      1      1
0      0      0      0      0      1
Tcycl =
1      1      0      0      0      0

```

```

1      1      0      0      0      0
0      0      1      0      0      0
0      0      0      1      0      0
0      0      0      0      1      0
0      0      0      0      0      1
TR =
1      1      1      0      0      0
1      1      1      1      1      1
0      0      1      1      1      1
0      0      0      1      1      1
0      0      0      0      1      1
0      0      0      0      0      1
TS =
1      1      1      0      0      0
1      1      1      1      1      1
1      1      1      1      1      1
0      1      1      1      1      1
0      1      1      1      1      1
0      1      1      1      1      1
MATLAB implementation of relational algebra boolean matrix operations in inline functions
TSn =
1      1      1      1      1      1
1      1      1      1      1      1
1      1      1      1      1      1
1      1      1      1      1      1
1      1      1      1      1      1
1      1      1      1      1      1
TSplus =
1      1      1      1      1      1
1      1      1      1      1      1
1      1      1      1      1      1
1      1      1      1      1      1
1      1      1      1      1      1
1      1      1      1      1      1
TX =
0      1      1      0      0      0
1      0      1      1      1      1
0      0      0      1      1      1
0      0      0      0      1      1
0      0      0      0      0      1
0      0      0      0      0      0
T =
0      1      1      0      0      0
0      0      1      1      1      1
0      0      0      1      1      1
0      0      0      0      1      1
0      0      0      0      0      1
0      0      0      0      0      0
Topological Sort: Successor determination completed
level =
2
Topological Sort: Successor determination completed
level =
3
Topological Sort: Successor determination completed
level =
4
Topological Sort: Successor determination completed
level =
5
Topological Sort: Successor determination completed
level =
6
Topological Sort: Successor determination completed
level =
7
Topological Sort: Logical taskschedule computed
TaskSchedule =
1      0      0      0      0      0
0      1      0      0      0      0
0      0      1      0      0      0

```

```

0      0      0      1      0      0
0      0      0      0      1      0
0      0      0      0      0      1
Taskschedule =
1      0      0      0      0      0
0      1      0      0      0      0
0      0      1      0      0      0
0      0      0      1      0      0
0      0      0      0      1      0
0      0      0      0      0      1
PavailableWithP =
0      0      0      0      0
0      1      0      0      0
0      0      1      1      0
0      0      1      1      0
0      0      0      0      1
isHomog =
1
vertexLabels =
Columns 1 through 4
'Client Owner'      'Architect'      'Engineer'      'Quantity Surveyor'
Columns 5 through 6
[1x30 char]      [1x30 char]
File YPavailableWithP.tgf opened
nRows =
5
nCols =
5
File YPavailableWithP.tgf closed
PtestunionP =
1      1      1      1      1
1      1      1      1      1
1      1      1      1      1
1      1      1      1      1
1      1      1      1      1
isHomog =
1
vertexLabels =
Columns 1 through 4
'Client Owner'      'Architect'      'Engineer'      'Quantity Surveyor'
Columns 5 through 6
[1x30 char]      [1x31 char]
File YPtestunionP.tgf opened
nRows =
5
nCols =
5
File YPtestunionP.tgf closed
TavailableWithT =
0      0      0      0      0      0
0      1      0      0      0      0
0      0      1      0      0      0
0      0      0      1      0      0
0      0      0      0      1      0
0      0      0      0      0      1
isHomog =
1
vertexLabels =
Columns 1 through 4
'Conceptualise'      'Plan'      'Eng Design'      'Specify & Document'
Columns 5 through 7
'Take off Quantities'      'Build & Construct'      [1x26 char]
File YTavailableWithT.tgf opened
nRows =
6
nCols =
6
File YTavailableWithT.tgf closed
TtestunionT =
1      1      1      1      1      1
1      1      1      1      1      1

```

1	1	1	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1

G.2 MATLAB Relational Algebra Functionality

The MATLAB code for the basic relational algebra functionality is listed below.

```
%%
% .....
%
%   Relational Algebra – Boolean matrix representation
%   MATLAB Matrix operations
% .....
%%
display(strcat('MATLAB implementation of relational algebra', ...
               ' boolean matrix operations in inline functions'))
% .....
%%
% zero, one and identity
zeroB=inline('logical(zeros(n,m))','n','m'); % e.g. Z4=zeroB(4,4)
oneB=inline('logical(ones(n,m))','n','m'); % e.g. Z4=zeroB(4,4)
identityB=inline('logical(eye(n))','n'); % e.g. I3=identityB(3)
%%
% .....
% Basic relational operations
productR= ...
inline('logical(mod(ones(size(fix(x)*fix(y))), (fix(x)*fix(y))+1))','x','y');
% Note: MATLAB standard set union is a=union([1,1],[1,1])
unionR= ...
inline('logical(mod(ones(size(x)), (x+y)+1))','x','y') ;
intersectionR= ...
inline('logical(mod(ones(size(x)), (x.*y)+1))','x','y');
differenceR=inline( ...
    'logical(mod(ones(size(x)), (x-mod(ones(size(x)), (x.*y)+1))+1))','x','y');
complementR=inline('logical(zeros(size(x))+not(logical(x)))','x');
transposeR=inline('transpose(x)','x');
% .....
%%
```

G.3 Process model database output MATLAB function

The MATLAB code to output the process model data to .csv file for importing into the engineering process model database is listed below.

```
function []=databaseDataWrite( ...
    taskLabels, personLabels, datasetLabels, toolLabels, ...
    statusElementLabels, stepLabels, ...
    attributeLabels, attributeDiscipline, ...
    attributeFunction, attributeType, ...
    edgeTaskPerson, edgeTaskTool, ...
    edgeDatasetTool, ...
    edgeTaskDatasetCreate, ...
    statusTaskDatasetCreate, ...
    edgeTaskDatasetRead, ...
    edgeTaskDatasetModify, ...
    statusTaskDatasetModify, ...
    weightDataset, ...
    statusCompletion, ...
    statusTaskDataset, ...)
```

```

    edgeTaskTask, taskStep, filePreFix)
% .....
% databaseDataWrite.m
% Output database data in .csv files
% Database design for table specifications
% .....
%
% Output person table
nPerson=size(personLabels,2)
tbPerson={};
for iPerson=1:nPerson
    tbPerson{iPerson,1}=strcat('P',num2str(iPerson,'%04d'));
    tbPerson{iPerson,2}=personLabels{iPerson};
end
CellWrite(strcat(filePreFix,'tbPerson.csv'),tbPerson)
%
% Output task table
nTask=size(taskLabels,2)
tbTask={};
for iTask=1:nTask
    tbTask{iTask,1}=strcat('T',num2str(iTask,'%04d'));
    tbTask{iTask,2}=taskLabels{iTask};
% use Stepname supplied
    tbTask{iTask,3}=strcat(stepLabels{ ...
        find(taskStep(iTask,:))});
%     tbTask{iTask,3}=strcat('Step', ...
%         num2str(find(taskStep(iTask,:)),'%04d'));
end
CellWrite(strcat(filePreFix,'tbTask.csv'),tbTask)
%
% Output tool table
nTool=size(toolLabels,2)
tbTool={};
for iTool=1:nTool
    tbTool{iTool,1}=strcat('G',num2str(iTool,'%04d'));
    tbTool{iTool,2}=toolLabels{iTool};
end
CellWrite(strcat(filePreFix,'tbTool.csv'),tbTool)
%
% Output dataset table
nDataset=size(datasetLabels,2)
tbDataset={};
for iDataset=1:nDataset
    tbDataset{iDataset,1}=strcat('D',num2str(iDataset,'%04d'));
    tbDataset{iDataset,2}=datasetLabels{iDataset};
    % Attribute & Status null
    tbDataset{iDataset,3}='';
    tbDataset{iDataset,4}='';
    tbDataset{iDataset,5}=num2str(weightDataset(iDataset),'%3.2f');
end
CellWrite(strcat(filePreFix,'tbDataset.csv'),tbDataset)
%
% Output statusElement table
nStatusElement=size(statusElementLabels,2)
tbStatusElement={};
for iStatusElement=1:nStatusElement
    tbStatusElement{iStatusElement,1}= ...
        strcat('S',num2str(iStatusElement,'%04d'));
    tbStatusElement{iStatusElement,2}= ...
        statusElementLabels{iStatusElement};
    tbStatusElement{iStatusElement,3}= ...
        num2str(statusCompletion(iStatusElement),'%3.2f');
end
CellWrite(strcat(filePreFix,'tbStatusElement.csv'),tbStatusElement)
%
% Output attribute table
nAttribute=size(attributeLabels,2)
tbAttribute={};
for iAttribute=1:nAttribute
    tbAttribute{iAttribute,1}= ...
        strcat('A',num2str(iAttribute,'%04d'));

```



```

        tbAttribute{iAttribute,2}= ...
            attributeLabels{iAttribute};
        tbAttribute{iAttribute,3}= ...
            attributeDiscipline{iAttribute};
        tbAttribute{iAttribute,4}= ...
            attributeFunction{iAttribute};
        tbAttribute{iAttribute,5}= ...
            attributeType{iAttribute};
    end
    CellWrite(strcat(filePreFix,'tbAttribute.csv'),tbAttribute)
%
% Output edgeTaskPerson table
nTaskE=size(edgeTaskPerson,1)
nPersonE=size(edgeTaskPerson,2)
if(nTaskE ~= nTask)
    disp(['Task label/edge count inconsistent nTaskE, nTask: ', ...
        num2str(nTaskE), ' ', num2str(nTask)])
end
if(nPersonE ~= nPerson)
    disp(['Person label/edge count inconsistent nPersonE, nPerson: ', ...
        num2str(nPersonE), ' ', num2str(nPerson)])
end
tbEdgeTaskPerson={};
iEdge=0;
for iTask=1:nTask
    for iPerson=1:nPerson
        if not(edgeTaskPerson(iTask,iPerson) == 0)
            iEdge=iEdge+1;
            tbEdgeTaskPerson{iEdge,1}= ...
                strcat('T',num2str(iTask,'%04d'),...
                    'P',num2str(iPerson,'%04d'));
            tbEdgeTaskPerson{iEdge,2}= ...
                strcat(char(taskLabels{iTask}),'-', ...
                    char(personLabels{iPerson}));
            tbEdgeTaskPerson{iEdge,3}= ...
                strcat('T',num2str(iTask,'%04d'));
            tbEdgeTaskPerson{iEdge,4}= ...
                strcat('P',num2str(iPerson,'%04d'));
            % no attribute and status element stored at present !!!!!!!!!!!!!!!
            tbEdgeTaskPerson{iEdge,5}= ' ';
            tbEdgeTaskPerson{iEdge,6}= ' ';
        end
    end
end
disp(['Edges Processed = ',num2str(iEdge)])
CellWrite(strcat(filePreFix,'tbEdgeTaskPerson.csv'),tbEdgeTaskPerson)
%
% Output edgeTaskTool table
nTaskE=size(edgeTaskTool,1)
nToolE=size(edgeTaskTool,2)
if(nTaskE ~= nTask)
    disp(['Task label/edge count inconsistent nTaskE, nTask: ', ...
        num2str(nTaskE), ' ', num2str(nTask)])
end
if(nToolE ~= nTool)
    disp(['Tool label/edge count inconsistent nToolE, nTool: ', ...
        num2str(nToolE), ' ', num2str(nTool)])
end
tbEdgeTaskTool={};
iEdge=0;
for iTask=1:nTask
    for iTool=1:nTool
        if not(edgeTaskTool(iTask,iTool) == 0)
            iEdge=iEdge+1;
            tbEdgeTaskTool{iEdge,1}= ...
                strcat('T',num2str(iTask,'%04d'),...
                    'G',num2str(iTool,'%04d'));
            tbEdgeTaskTool{iEdge,2}= ...
                strcat(char(taskLabels{iTask}),'-', ...
                    char(toolLabels{iTool}));
            tbEdgeTaskTool{iEdge,3}= ...

```

```

        strcat('T', num2str(iTask, '%04d'));
        tbEdgeTaskTool{iEdge, 4} = ...
        strcat('G', num2str(iTool, '%04d'));
% no attribute and status element stored at present !!!!!!!!!!!!!!!
        tbEdgeTaskTool{iEdge, 5} = ' ';
        tbEdgeTaskTool{iEdge, 6} = ' ';
    end
end
end
disp(['Edges Processed = ', num2str(iEdge)])
CellWrite(strcat(filePreFix, 'tbEdgeTaskTool.csv'), tbEdgeTaskTool)
%
% Output edgeDatasetTool table
nDatasetE = size(edgeDatasetTool, 1)
nToolE = size(edgeDatasetTool, 2)
if(nDatasetE ~= nDataset)
    disp(['Dataset label/edge count inconsistent nDatasetE, nDataset: ', ...
        num2str(nDatasetE), ' ', num2str(nDataset)])
end
if(nToolE ~= nTool)
    disp(['Tool label/edge count inconsistent nToolE, nTool: ', ...
        num2str(nToolE), ' ', num2str(nTool)])
end
tbEdgeDatasetTool = {};
iEdge = 0;
for iDataset = 1:nDataset
    for iTool = 1:nTool
        if not(edgeDatasetTool(iDataset, iTool) == 0)
            iEdge = iEdge + 1;
            tbEdgeDatasetTool{iEdge, 1} = ...
                strcat('D', num2str(iDataset, '%04d'), ...
                    'G', num2str(iTool, '%04d'));
            tbEdgeDatasetTool{iEdge, 2} = ...
                strcat(char(datasetLabels{iDataset}), '-', ...
                    char(toolLabels{iTool}));
            tbEdgeDatasetTool{iEdge, 3} = ...
                strcat('D', num2str(iDataset, '%04d'));
            tbEdgeDatasetTool{iEdge, 4} = ...
                strcat('G', num2str(iTool, '%04d'));
% no attribute and status element stored at present !!!!!!!!!!!!!!!
            tbEdgeDatasetTool{iEdge, 5} = ' ';
            tbEdgeDatasetTool{iEdge, 6} = ' ';
        end
    end
end
disp(['Edges Processed = ', num2str(iEdge)])
CellWrite(strcat(filePreFix, 'tbEdgeDatasetTool.csv'), tbEdgeDatasetTool)
%
% Output edgeTaskDatasetCreate table without status values
nTaskE = size(edgeTaskDatasetCreate, 1)
nDatasetE = size(edgeTaskDatasetCreate, 2)
nDatasetStatusElement = size(statusElementLabels, 2)
if(nTaskE ~= nTask)
    disp(['Task label/edge count inconsistent nTaskE, nTask: ', ...
        num2str(nTaskE), ' ', num2str(nTask)])
end
if(nDatasetE ~= nDataset)
    disp(['Dataset label/edge count inconsistent nDatasetE, nDataset: ', ...
        num2str(nDatasetE), ' ', num2str(nDataset)])
end
tbEdgeTaskDatasetCreate = {};
iEdge = 0;
for iTask = 1:nTaskE
    for iDataset = 1:nDataset
        if not(edgeTaskDatasetCreate(iTask, iDataset) == 0)
            iEdge = iEdge + 1;
            tbEdgeTaskDatasetCreate{iEdge, 1} = ...
                strcat('T', num2str(iTask, '%04d'), ...
                    'D', num2str(iDataset, '%04d'));
            tbEdgeTaskDatasetCreate{iEdge, 2} = ...

```

```

        strcat(char(taskLabels{iTask}),'-', ...
               char(datasetLabels{iDataset}));
    tbEdgeTaskDatasetCreate{iEdge,3}= ...
        strcat('T',num2str(iTask,'%04d'));
    tbEdgeTaskDatasetCreate{iEdge,4}= ...
        strcat('D',num2str(iDataset,'%04d'));
% no attribute stored at present !!!!!!!!!!!!!!!
    tbEdgeTaskDatasetCreate{iEdge,5}= ' ';
% store create status element for task - data edge
    PIDStatusElement='';
    if(nDatasetStatusElement >0)
    iStatusElement=statusTaskDatasetCreate(iTask,iDataset);
        if iStatusElement >0
            PIDStatusElement=strcat('S',num2str(iStatusElement,'%04d'));
        end
    end
    tbEdgeTaskDatasetCreate{iEdge,6}=PIDStatusElement;
end
end
end
disp(['Edges Processed = ',num2str(iEdge)])
CellWrite(strcat(filePrefix,'tbEdgeTaskDatasetCreate.csv'), ...
          tbEdgeTaskDatasetCreate)

%
% Output edgeTaskDatasetRead table without status values
nTaskE=size(edgeTaskDatasetRead,1)
nDatasetE=size(edgeTaskDatasetRead,2)
if(nTaskE ~= nTask)
    disp(['Task label/edge count inconsistent nTaskE, nTask: ', ...
          num2str(nTaskE), ' ',num2str(nTask)])
end
if(nDatasetE ~= nDataset)
    disp(['Dataset label/edge count inconsistent nDatasetE, nDataset: ', ...
          num2str(nDatasetE), ' ',num2str(nDataset)])
end
end
tbEdgeTaskDatasetRead={};
iEdge=0;
for iTask=1:nTaskE
    for iDataset=1:nDataset
        if not(edgeTaskDatasetRead(iTask,iDataset) == 0)
            iEdge=iEdge+1;
            tbEdgeTaskDatasetRead{iEdge,1}= ...
                strcat('T',num2str(iTask,'%04d'),...
                       'D',num2str(iDataset,'%04d'));
            tbEdgeTaskDatasetRead{iEdge,2}= ...
                strcat(char(taskLabels{iTask}),'-', ...
                       char(datasetLabels{iDataset}));
            tbEdgeTaskDatasetRead{iEdge,3}= ...
                strcat('T',num2str(iTask,'%04d'));
            tbEdgeTaskDatasetRead{iEdge,4}= ...
                strcat('D',num2str(iDataset,'%04d'));
% no attribute stored at present !!!!!!!!!!!!!!!
            tbEdgeTaskDatasetRead{iEdge,5}= ' ';
            tbEdgeTaskDatasetRead{iEdge,6}= ' ';
        end
    end
end
disp(['Edges Processed = ',num2str(iEdge)])
CellWrite(strcat(filePrefix,'tbEdgeTaskDatasetRead.csv'), ...
          tbEdgeTaskDatasetRead)

%
% Output edgeTaskDatasetModify table with status values
nTaskE=size(edgeTaskDatasetModify,1)
nDatasetE=size(edgeTaskDatasetModify,2)
nDatasetStatus=size(statusTaskDataset,2)
nDatasetStatusElement=size(statusElementLabels,2)
if(nTaskE ~= nTask)
    disp(['Task label/edge count inconsistent nTaskE, nTask: ', ...
          num2str(nTaskE), ' ',num2str(nTask)])

```

```

end
if (nDatasetE ~= nDataset)
    disp(['Dataset label/edge count inconsistent nDatasetE, nDataset: ', ...
        num2str(nDatasetE), ' ', num2str(nDataset)])
end

if (nDatasetStatus >= 0)
    if (nDatasetE ~= nDatasetStatus)
        disp(['Dataset label/status array count inconsistent nDatasetE, ', ...
            'nDatasetStatus: ', ...
            num2str(nDatasetE), ' ', num2str(nDatasetStatus)])
    end
end

end
end
tbEdgeTaskDatasetModify = {};
iEdge = 0;
for iTask = 1:nTaskE
    for iDataset = 1:nDataset
        if not (edgeTaskDatasetModify(iTask, iDataset) == 0)
            iEdge = iEdge + 1;
            tbEdgeTaskDatasetModify{iEdge, 1} = ...
                strcat('T', num2str(iTask, '%04d'), ...
                    'D', num2str(iDataset, '%04d'));
            tbEdgeTaskDatasetModify{iEdge, 2} = ...
                strcat(char(taskLabels{iTask}), '-', ...
                    char(datasetLabels{iDataset}));
            tbEdgeTaskDatasetModify{iEdge, 3} = ...
                strcat('T', num2str(iTask, '%04d'));
            tbEdgeTaskDatasetModify{iEdge, 4} = ...
                strcat('D', num2str(iDataset, '%04d'));
            % no Attribute stored at present !!!!!!!!!!!!!!!
            tbEdgeTaskDatasetModify{iEdge, 5} = ' ';
            % store modify status element fore task - data edge
            PIDStatusElement = '';
            if (nDatasetStatusElement > 0)
                iStatusElement = statusTaskDatasetModify(iTask, iDataset);
                if iStatusElement > 0
                    PIDStatusElement = strcat('S', num2str(iStatusElement, '%04d'));
                end
            end
            tbEdgeTaskDatasetModify{iEdge, 6} = PIDStatusElement;
        end
    end
end

end
disp(['Edges Processed = ', num2str(iEdge)])
CellWrite(strcat(filePreFix, 'tbEdgeTaskDatasetModify.csv'), ...
    tbEdgeTaskDatasetModify)
%
% Output edgeTaskTask table
nTaskERow = size(edgeTaskTask, 1)
nTaskECol = size(edgeTaskTask, 2)
if (nTaskERow ~= nTask)
    disp(['Task label/edge count inconsistent nTaskERow, nTask: ', ...
        num2str(nTaskERow), ' ', num2str(nTask)])
end

if (nTaskECol ~= nTask)
    disp(['Task label/edge count inconsistent nTaskECol, nTask: ', ...
        num2str(nTaskECol), ' ', num2str(nTask)])
end

end
tbEdgeTaskTask = {};
iEdge = 0;
for iTaskRow = 1:nTaskERow
    for iTaskCol = 1:nTaskECol
        if not (edgeTaskTask(iTaskRow, iTaskCol) == 0)
            iEdge = iEdge + 1;
            tbEdgeTaskTask{iEdge, 1} = ...
                strcat('T', num2str(iTaskRow, '%04d'), ...
                    'T', num2str(iTaskCol, '%04d'));
            tbEdgeTaskTask{iEdge, 2} = ...
                strcat(char(taskLabels{iTaskRow}), '-', ...
                    char(taskLabels{iTaskCol}));
        end
    end
end

```

```

        tbEdgeTaskTask{iEdge,3}= ...
            strcat('T',num2str(iTaskRow,'%04d'));
        tbEdgeTaskTask{iEdge,4}= ...
            strcat('T',num2str(iTaskCol,'%04d'));
% no attribute and status element stored at present !!!!!!!!!!!!!!!
        tbEdgeTaskTask{iEdge,5}= ' ';
        tbEdgeTaskTask{iEdge,6}= ' ';
        tbEdgeTaskTask{iEdge,7}= ...
            strcat('T',num2str(iTaskRow,'%04d'));
    end
end
end
disp(['Edges Processed = ',num2str(iEdge)])
CellWrite(strcat(filePrefix,'tbEdgeTaskTask.csv'),tbEdgeTaskTask)

```

G.4 Graph data formats used by the yEd program

Examples of the .tgf and .xgml data file formats used by the yEd program are listed below.

YPexecutesT.tgf

```

1 Client Owner
2 Architect
3 Engineer
4 Quantity Surveyor
5 Constructor/Contractor/Builder
6 Conceptualise
7 Plan
8 Eng Design
9 Specify & Document
10 Take off Quantities
11 Build & Construct
12 Person executes Task
#
1 6
2 6
2 7
3 8
3 9
4 9
4 10
5 11

```

YPexecutesT.xgml

```

<section name="xgml">
<attribute key="Creator" type="String">yFiles</attribute>
<attribute key="Version" type="String">2.3.1</attribute>
<section name="graph">
<attribute key="hierarchic" type="int">1</attribute>
<attribute key="label" type="String"></attribute>
<attribute key="directed" type="int">1</attribute>
<section name="node">
<attribute key="id" type="int">0</attribute>
<attribute key="label" type="String">Client Owner</attribute>
<section name="graphics">
<attribute key="x" type="double">-190.0</attribute>
<attribute key="y" type="double">12.0</attribute>
<attribute key="w" type="double">88.0</attribute>
<attribute key="h" type="double">30.0</attribute>
<attribute key="type" type="String">rectangle</attribute>
<attribute key="fill" type="String">#FFCC99</attribute>
<attribute key="outline" type="String">#000000</attribute>
</section>
<section name="LabelGraphics">
<attribute key="text" type="String">Client Owner</attribute>
<attribute key="fontSize" type="int">13</attribute>
<attribute key="fontName" type="String">Dialog</attribute>
<attribute key="anchor" type="String">c</attribute>

```

```

</section>
</section>
<section name="node">
<attribute key="id" type="int">1</attribute>
<attribute key="label" type="String">Architect</attribute>
<section name="graphics">
<attribute key="x" type="double">-93.0</attribute>
<attribute key="y" type="double">12.0</attribute>
<attribute key="w" type="double">66.0</attribute>
<attribute key="h" type="double">30.0</attribute>
<attribute key="type" type="String">rectangle</attribute>
<attribute key="fill" type="String">#FFCC99</attribute>
<attribute key="outline" type="String">#000000</attribute>
</section>
<section name="LabelGraphics">
<attribute key="text" type="String">Architect</attribute>
<attribute key="fontSize" type="int">13</attribute>
<attribute key="fontName" type="String">Dialog</attribute>
<attribute key="anchor" type="String">c</attribute>
</section>
</section>
<section name="node">
<attribute key="id" type="int">2</attribute>
<attribute key="label" type="String">Conceptualise</attribute>
<section name="graphics">
<attribute key="x" type="double">-166.0</attribute>
<attribute key="y" type="double">82.0</attribute>
<attribute key="w" type="double">96.0</attribute>
<attribute key="h" type="double">30.0</attribute>
<attribute key="type" type="String">rectangle</attribute>
<attribute key="fill" type="String">#FFCC00</attribute>
<attribute key="outline" type="String">#000000</attribute>
</section>
<section name="LabelGraphics">
<attribute key="text" type="String">Conceptualise</attribute>
<attribute key="fontSize" type="int">13</attribute>
<attribute key="fontName" type="String">Dialog</attribute>
<attribute key="anchor" type="String">c</attribute>
</section>
</section>
<section name="node">
<attribute key="id" type="int">3</attribute>
<attribute key="label" type="String">Plan</attribute>
<section name="graphics">
<attribute key="x" type="double">-76.5</attribute>
<attribute key="y" type="double">82.0</attribute>
<attribute key="w" type="double">40.0</attribute>
<attribute key="h" type="double">30.0</attribute>
<attribute key="type" type="String">rectangle</attribute>
<attribute key="fill" type="String">#FFCC00</attribute>
<attribute key="outline" type="String">#000000</attribute>
</section>
<section name="LabelGraphics">
<attribute key="text" type="String">Plan</attribute>
<attribute key="fontSize" type="int">13</attribute>
<attribute key="fontName" type="String">Dialog</attribute>
<attribute key="anchor" type="String">c</attribute>
</section>
</section>
<section name="node">
<attribute key="id" type="int">4</attribute>
<attribute key="label" type="String">Person executes Task</attribute>
<section name="graphics">
<attribute key="x" type="double">-160.5</attribute>
<attribute key="y" type="double">-45.0</attribute>
<attribute key="w" type="double">144.0</attribute>
<attribute key="h" type="double">30.0</attribute>
<attribute key="type" type="String">rectangle</attribute>
<attribute key="fill" type="String">#99CC00</attribute>
<attribute key="outline" type="String">#000000</attribute>
</section>
<section name="LabelGraphics">
<attribute key="text" type="String">Person executes Task</attribute>
<attribute key="fontSize" type="int">13</attribute>
<attribute key="fontName" type="String">Dialog</attribute>
<attribute key="anchor" type="String">c</attribute>
</section>
</section>
<section name="node">
<attribute key="id" type="int">5</attribute>
<attribute key="label" type="String">Build & Construct</attribute>
<section name="graphics">
<attribute key="x" type="double">437.5</attribute>
<attribute key="y" type="double">80.0</attribute>
<attribute key="w" type="double">116.0</attribute>

```

```

<attribute key="h" type="double">30.0</attribute>
<attribute key="type" type="String">rectangle</attribute>
<attribute key="fill" type="String">#FFCC00</attribute>
<attribute key="outline" type="String">#000000</attribute>
</section>
<section name="LabelGraphics">
<attribute key="text" type="String">Build & Construct</attribute>
<attribute key="fontSize" type="int">13</attribute>
<attribute key="fontName" type="String">Dialog</attribute>
<attribute key="anchor" type="String">c</attribute>
</section>
</section>
<section name="node">
<attribute key="id" type="int">6</attribute>
<attribute key="label" type="String">Take off Quantities</attribute>
<section name="graphics">
<attribute key="x" type="double">23.0</attribute>
<attribute key="y" type="double">82.0</attribute>
<attribute key="w" type="double">122.0</attribute>
<attribute key="h" type="double">30.0</attribute>
<attribute key="type" type="String">rectangle</attribute>
<attribute key="fill" type="String">#FFCC00</attribute>
<attribute key="outline" type="String">#000000</attribute>
</section>
<section name="LabelGraphics">
<attribute key="text" type="String">Take off Quantities</attribute>
<attribute key="fontSize" type="int">13</attribute>
<attribute key="fontName" type="String">Dialog</attribute>
<attribute key="anchor" type="String">c</attribute>
</section>
</section>
<section name="node">
<attribute key="id" type="int">7</attribute>
<attribute key="label" type="String">Specify & Document</attribute>
<section name="graphics">
<attribute key="x" type="double">170.5</attribute>
<attribute key="y" type="double">82.0</attribute>
<attribute key="w" type="double">133.0</attribute>
<attribute key="h" type="double">30.0</attribute>
<attribute key="type" type="String">rectangle</attribute>
<attribute key="fill" type="String">#FFCC00</attribute>
<attribute key="outline" type="String">#000000</attribute>
</section>
<section name="LabelGraphics">
<attribute key="text" type="String">Specify & Document</attribute>
<attribute key="fontSize" type="int">13</attribute>
<attribute key="fontName" type="String">Dialog</attribute>
<attribute key="anchor" type="String">c</attribute>
</section>
</section>
<section name="node">
<attribute key="id" type="int">8</attribute>
<attribute key="label" type="String">Eng Design</attribute>
<section name="graphics">
<attribute key="x" type="double">297.5</attribute>
<attribute key="y" type="double">82.0</attribute>
<attribute key="w" type="double">81.0</attribute>
<attribute key="h" type="double">30.0</attribute>
<attribute key="type" type="String">rectangle</attribute>
<attribute key="fill" type="String">#FFCC00</attribute>
<attribute key="outline" type="String">#000000</attribute>
</section>
<section name="LabelGraphics">
<attribute key="text" type="String">Eng Design</attribute>
<attribute key="fontSize" type="int">13</attribute>
<attribute key="fontName" type="String">Dialog</attribute>
<attribute key="anchor" type="String">c</attribute>
</section>
</section>
<section name="node">
<attribute key="id" type="int">9</attribute>
<attribute key="label" type="String">Constructor / Contractor / Builder</attribute>
<section name="graphics">
<attribute key="x" type="double">437.5</attribute>
<attribute key="y" type="double">10.0</attribute>
<attribute key="w" type="double">189.0</attribute>
<attribute key="h" type="double">30.0</attribute>
<attribute key="type" type="String">rectangle</attribute>
<attribute key="fill" type="String">#FFCC99</attribute>
<attribute key="outline" type="String">#000000</attribute>
</section>
<section name="LabelGraphics">
<attribute key="text" type="String">Constructor / Contractor / Builder</attribute>
<attribute key="fontSize" type="int">13</attribute>
<attribute key="fontName" type="String">Dialog</attribute>

```

```

<attribute key="anchor" type="String">c</attribute>
</section>
</section>
<section name="node">
<attribute key="id" type="int">10</attribute>
<attribute key="label" type="String">Quantity Surveyor</attribute>
<section name="graphics">
<attribute key="x" type="double">80.125</attribute>
<attribute key="y" type="double">12.0</attribute>
<attribute key="w" type="double">117.0</attribute>
<attribute key="h" type="double">30.0</attribute>
<attribute key="type" type="String">rectangle</attribute>
<attribute key="fill" type="String">#FFCC99</attribute>
<attribute key="outline" type="String">#000000</attribute>
</section>
<section name="LabelGraphics">
<attribute key="text" type="String">Quantity Surveyor</attribute>
<attribute key="fontSize" type="int">13</attribute>
<attribute key="fontName" type="String">Dialog</attribute>
<attribute key="anchor" type="String">c</attribute>
</section>
</section>
<section name="node">
<attribute key="id" type="int">11</attribute>
<attribute key="label" type="String">Engineer</attribute>
<section name="graphics">
<attribute key="x" type="double">250.625</attribute>
<attribute key="y" type="double">12.0</attribute>
<attribute key="w" type="double">65.0</attribute>
<attribute key="h" type="double">30.0</attribute>
<attribute key="type" type="String">rectangle</attribute>
<attribute key="fill" type="String">#FFCC99</attribute>
<attribute key="outline" type="String">#000000</attribute>
</section>
<section name="LabelGraphics">
<attribute key="text" type="String">Engineer</attribute>
<attribute key="fontSize" type="int">13</attribute>
<attribute key="fontName" type="String">Dialog</attribute>
<attribute key="anchor" type="String">c</attribute>
</section>
</section>
<section name="edge">
<attribute key="source" type="int">0</attribute>
<attribute key="target" type="int">2</attribute>
<section name="graphics">
<attribute key="fill" type="String">#000000</attribute>
<attribute key="targetArrow" type="String">standard</attribute>
</section>
<section name="edgeAnchor">
<attribute key="ySource" type="double">1.0</attribute>
<attribute key="xTarget" type="double">-0.5</attribute>
<attribute key="yTarget" type="double">-1.0</attribute>
</section>
</section>
<section name="edge">
<attribute key="source" type="int">1</attribute>
<attribute key="target" type="int">2</attribute>
<section name="graphics">
<attribute key="fill" type="String">#000000</attribute>
<attribute key="targetArrow" type="String">standard</attribute>
</section>
<section name="Line">
<section name="point">
<attribute key="x" type="double">-93.0</attribute>
<attribute key="y" type="double">12.0</attribute>
</section>
<section name="point">
<attribute key="x" type="double">-109.5</attribute>
<attribute key="y" type="double">47.0</attribute>
</section>
<section name="point">
<attribute key="x" type="double">-142.0</attribute>
<attribute key="y" type="double">47.0</attribute>
</section>
<section name="point">
<attribute key="x" type="double">-166.0</attribute>
<attribute key="y" type="double">82.0</attribute>
</section>
</section>
</section>
<section name="edgeAnchor">
<attribute key="xSource" type="double">-0.5</attribute>
<attribute key="ySource" type="double">1.0</attribute>
<attribute key="xTarget" type="double">0.5</attribute>
<attribute key="yTarget" type="double">-1.0</attribute>
</section>

```



```

</section>
<section name="edge">
<attribute key="source" type="int">1</attribute>
<attribute key="target" type="int">3</attribute>
<section name="graphics">
<attribute key="fill" type="String">#000000</attribute>
<attribute key="targetArrow" type="String">standard</attribute>
</section>
<section name="edgeAnchor">
<attribute key="xSource" type="double">0.5</attribute>
<attribute key="ySource" type="double">1.0</attribute>
<attribute key="yTarget" type="double">-1.0</attribute>
</section>
</section>
<section name="edge">
<attribute key="source" type="int">11</attribute>
<attribute key="target" type="int">8</attribute>
<section name="graphics">
<attribute key="fill" type="String">#000000</attribute>
<attribute key="targetArrow" type="String">standard</attribute>
<section name="Line">
<section name="point">
<attribute key="x" type="double">250.625</attribute>
<attribute key="y" type="double">12.0</attribute>
</section>
<section name="point">
<attribute key="x" type="double">266.875</attribute>
<attribute key="y" type="double">47.0</attribute>
</section>
<section name="point">
<attribute key="x" type="double">297.5</attribute>
<attribute key="y" type="double">47.0</attribute>
</section>
<section name="point">
<attribute key="x" type="double">297.5</attribute>
<attribute key="y" type="double">82.0</attribute>
</section>
</section>
<section name="edgeAnchor">
<attribute key="xSource" type="double">0.5</attribute>
<attribute key="ySource" type="double">1.0</attribute>
<attribute key="yTarget" type="double">-1.0</attribute>
</section>
</section>
<section name="edge">
<attribute key="source" type="int">11</attribute>
<attribute key="target" type="int">7</attribute>
<section name="graphics">
<attribute key="fill" type="String">#000000</attribute>
<attribute key="targetArrow" type="String">standard</attribute>
<section name="Line">
<section name="point">
<attribute key="x" type="double">250.625</attribute>
<attribute key="y" type="double">12.0</attribute>
</section>
<section name="point">
<attribute key="x" type="double">234.375</attribute>
<attribute key="y" type="double">47.0</attribute>
</section>
<section name="point">
<attribute key="x" type="double">203.75</attribute>
<attribute key="y" type="double">47.0</attribute>
</section>
<section name="point">
<attribute key="x" type="double">170.5</attribute>
<attribute key="y" type="double">82.0</attribute>
</section>
</section>
</section>
<section name="edgeAnchor">
<attribute key="xSource" type="double">-0.5</attribute>
<attribute key="ySource" type="double">1.0</attribute>
<attribute key="xTarget" type="double">0.5</attribute>
<attribute key="yTarget" type="double">-1.0</attribute>
</section>
</section>
<section name="edge">
<attribute key="source" type="int">10</attribute>
<attribute key="target" type="int">7</attribute>
<section name="graphics">
<attribute key="fill" type="String">#000000</attribute>
<attribute key="targetArrow" type="String">standard</attribute>
<section name="Line">
<section name="point">

```

```

<attribute key="x" type="double">80.125</attribute>
<attribute key="y" type="double">12.0</attribute>
</section>
<section name="point">
<attribute key="x" type="double">109.375</attribute>
<attribute key="y" type="double">47.0</attribute>
</section>
<section name="point">
<attribute key="x" type="double">137.25</attribute>
<attribute key="y" type="double">47.0</attribute>
</section>
<section name="point">
<attribute key="x" type="double">170.5</attribute>
<attribute key="y" type="double">82.0</attribute>
</section>
</section>
</section>
</section>
<section name="edgeAnchor">
<attribute key="xSource" type="double">0.5</attribute>
<attribute key="ySource" type="double">1.0</attribute>
<attribute key="xTarget" type="double">-0.5</attribute>
<attribute key="yTarget" type="double">-1.0</attribute>
</section>
</section>
<section name="edge">
<attribute key="source" type="int">10</attribute>
<attribute key="target" type="int">6</attribute>
<section name="graphics">
<attribute key="fill" type="String">#000000</attribute>
<attribute key="targetArrow" type="String">standard</attribute>
</section>
<section name="Line">
<section name="point">
<attribute key="x" type="double">80.125</attribute>
<attribute key="y" type="double">12.0</attribute>
</section>
<section name="point">
<attribute key="x" type="double">50.875</attribute>
<attribute key="y" type="double">47.0</attribute>
</section>
<section name="point">
<attribute key="x" type="double">23.0</attribute>
<attribute key="y" type="double">47.0</attribute>
</section>
<section name="point">
<attribute key="x" type="double">23.0</attribute>
<attribute key="y" type="double">82.0</attribute>
</section>
</section>
</section>
</section>
<section name="edgeAnchor">
<attribute key="xSource" type="double">-0.5</attribute>
<attribute key="ySource" type="double">1.0</attribute>
<attribute key="yTarget" type="double">-1.0</attribute>
</section>
</section>
<section name="edge">
<attribute key="source" type="int">9</attribute>
<attribute key="target" type="int">5</attribute>
<section name="graphics">
<attribute key="fill" type="String">#000000</attribute>
<attribute key="targetArrow" type="String">standard</attribute>
</section>
<section name="edgeAnchor">
<attribute key="ySource" type="double">1.0</attribute>
<attribute key="yTarget" type="double">-1.0</attribute>
</section>
</section>
</section>
</section>

```

G.5 Database file transfer format

The process model data in .csv file format is used for importing data into the engineering process model database.

The contents of two typical .csv files are listed below.

YEx1tbPerson.csv

```
P0001, Client Owner, A0003, S0005
P0002, Architect, A0003, S0005
P0003, Engineer, A0003, S0005
P0004, Quantity Surveyor, A0003, S0005
P0005, Constructor/ Contractor/ Builder, A0003, S0005
```

YEx1tbEdgeTaskPerson.csv

```
T0001P0001, Conceptualise-Client Owner, T0001, P0001, A0003, S0005
T0001P0002, Conceptualise-Architect, T0001, P0002, A0003, S0005
T0002P0002, Plan-Architect, T0002, P0002, A0003, S0005
T0003P0003, Eng Design-Engineer, T0003, P0003, A0003, S0005
T0004P0003, Specify & Document-Engineer, T0004, P0003, A0003, S0005
T0004P0004, Specify & Document-Quantity Surveyor, T0004, P0004, A0003, S0005
T0005P0004, Take off Quantities-Quantity Surveyor, T0005, P0004, A0003, S0005
T0006P0005, Build & Construct-Constructor/ Contractor/ Builder, T0006, P0005, A0003, S0005
```

G.6 MATLAB implementation of process model with status settings**Contents**

- Initialise relation algebra tools
- Set up labels for task and person data
- Process Person-task relations and output graph data
- Set up data and process dataset relations
- Compute tasks - data create relation
- Compute tasks - data read relation
- Compute tasks - data read relation
- Set up tool data and relations
- Person / Task / Data graph output
- Set up status information
- Solution
- Output task sequence graphs
- Combine task sequences and output task - step relation

```
%.....
% Process Model for typical Design Project - Example from Tutorial
% With status settings
% 2006/04/16
%.....
clc
clear all
format compact
```

Initialise relation algebra tools

```
RelationalAlgebraBoolean
```

```
MATLAB implementation of relational algebra boolean matrix operations in inline functions
```

Set up labels for task and person data

```

tgfFilePre='YS'
% Define persons executes tasks PT
% Tasks: ArchitecturalLayoutConcept / t1
%       ArchitecturalDesignDetail / t2
%       ArchitecturalDesignCheck / t3
%       StructuralLayoutConcept / t4
%       StructuralDesignDetail / t5
%       StructuralDesignCheck / t6
%       ConcreteLayoutConcept / t7
%       ConcreteDesignDetail / t8
%       ConcreteDesignCheck / t9
taskLabels= {'ArchitecturalLayoutConcept', ...
            'ArchitecturalDesignDetail ', ...
            'ArchitecturalDesignCheck', ...
            'StructuralLayoutConcept', ...
            'StructuralDesignDetail', ...
            'StructuralDesignCheck', ...
            'ConcreteLayoutConcept', ...
            'ConcreteDesignDetail', ...
            'ConcreteDesignCheck'}
personLabels={'Architect','Client','StructuralEngineer', ...
            'Technologist','CheckingEngineer'}

```

```

tgfFilePre =
YS
taskLabels =
Columns 1 through 5
[1x26 char] [1x26 char] [1x24 char] [1x23 char] [1x22 char]
Columns 6 through 9
[1x21 char] [1x21 char] [1x20 char] 'ConcreteDesignCheck'
personLabels =
Columns 1 through 4
'Architect' 'Client' 'StructuralEngineer' 'Technologist'
Column 5
'CheckingEngineer'

```

Process Person-task relations and output graph data

```

PexecutesT=([1 1 0 0 0 0 0 0 0 ; ... % Architect / p1
            0 0 0 0 0 0 0 0 0 ; ... % Client / p2
            0 0 0 1 1 0 0 1 0 ; ... % StructuralEngineer /p3
            0 0 0 0 0 0 1 1 0 ; ... % Technologist /p4
            0 0 1 0 0 1 0 0 1 ]) % CheckingEngineer /p5
% add comment & heading strings at end of labels
vertexLabels=horzcat(personLabels,taskLabels',{'Person executes Task'});
isHomog=0
filePT=horzcat(tgfFilePre,'PexecutesT',' .tgf')
tgfWrite(filePT,PexecutesT,isHomog,vertexLabels,{});
% Compute tasks - persons TP
vertexLabels=horzcat(taskLabels,personLabels',{'Task executed by Person'});
TexecutedByP=transposeR(PexecutesT)
isHomog=0
fileTP=horzcat(tgfFilePre,'TexecutedByP',' .tgf')
tgfWrite(fileTP,TexecutedByP,isHomog,vertexLabels,{});

```

```

PexecutesT =
1      1      0      0      0      0      0      0      0
0      0      0      0      0      0      0      0      0
0      0      0      1      1      0      0      1      0
0      0      0      0      0      0      1      1      0
0      0      1      0      0      1      0      0      1

isHomog =
0

```



```

YSDcreatedByT.tgf
File YSDcreatedByT.tgf opened
nRows =
    5
nCols =
    9
File YSDcreatedByT.tgf closed

```

Compute tasks - data create relation

```

vertexLabels=horzcat(taskLabels,dataLabels',{'Task creates Data'});
TcreatesD=transposeR(DcreatedByT)
isHomog=0
fileTcD=horzcat(tgfFilePre,'TcreatesD','.tgf')
tgfWrite(fileTcD,TcreatesD,isHomog,vertexLabels,{});
% Data is read by Task
DreadByT= ([ 1 0 0 0 0 0 0 0 0; ... % ConceptPlan
             0 1 1 1 0 0 1 0 0; ... % ArchitecturalDrawings
             0 0 0 0 1 1 1 0 0; ... % StructuralDrawings
             0 0 0 0 0 0 0 1 1; ... % ConcreteDrawings
             0 0 0 0 0 0 0 0 0]) % Debugger
% Output graph data
vertexLabels=horzcat(dataLabels,taskLabels',{'Data read by Task'});
isHomog=0
fileDrT=horzcat(tgfFilePre,'DreadByT','.tgf')
tgfWrite(fileDrT,DreadByT,isHomog,vertexLabels,{});

```

```

TcreatesD =
    0    1    0    0    0
    0    0    0    0    0
    0    0    0    0    0
    0    0    1    0    0
    0    0    0    0    0
    0    0    0    0    0
    0    0    0    1    0
    0    0    0    0    0
    0    0    0    0    0

isHomog =
    0
fileTcD =
YSTcreatesD.tgf
File YSTcreatesD.tgf opened
nRows =
    9
nCols =
    5
File YSTcreatesD.tgf closed
DreadByT =
    1    0    0    0    0    0    0    0    0
    0    1    1    1    0    0    1    0    0
    0    0    0    0    1    1    1    0    0
    0    0    0    0    0    0    0    1    1
    0    0    0    0    0    0    0    0    0

isHomog =
    0
fileDrT =
YSDreadByT.tgf
File YSDreadByT.tgf opened
nRows =
    5
nCols =
    9
File YSDreadByT.tgf closed

```

Compute tasks - data read relation

```

vertexLabels=horzcat(taskLabels,dataLabels',{'Task reads Data'});
TreadsD=transposeR(DreadByT)
isHomog=0
fileTrD=horzcat(tgfFilePre,'TreadsD','.tgf')
tgfWrite(fileTrD,TreadsD,isHomog,vertexLabels,{});
% Data is modified by Task
DmodifiedByT= ([ 0 0 0 0 0 0 0 0 0; ... % ConceptPlan
                 0 1 1 0 0 0 0 0 0; ... % ArchitecturalDrawings
                 0 0 0 0 1 1 0 0 0; ... % StructuralDrawings
                 0 0 0 0 0 0 0 1 1; ... % ConcreteDrawings
                 0 0 0 0 0 0 0 0 0]) % Debugger
% Output graph data
vertexLabels=horzcat(dataLabels,taskLabels',{'Data modified by Task'});
isHomog=0
fileDmT=horzcat(tgfFilePre,'DmodifiedByT','.tgf')
tgfWrite(fileDmT,DmodifiedByT,isHomog,vertexLabels,{});

```

```

TreadsD =
     1     0     0     0     0
     0     1     0     0     0
     0     1     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     1     0     0
     0     1     1     0     0
     0     0     0     1     0
     0     0     0     1     0

isHomog =
     0
fileTrD =
YSTreadsD.tgf
File YSTreadsD.tgf opened
nRows =
     9
nCols =
     5
File YSTreadsD.tgf closed
DmodifiedByT =
     0     0     0     0     0     0     0     0     0
     0     1     1     0     0     0     0     0     0
     0     0     0     0     0     1     1     0     0
     0     0     0     0     0     0     0     1     1
     0     0     0     0     0     0     0     0     0

isHomog =
     0
fileDmT =
YSDmodifiedByT.tgf
File YSDmodifiedByT.tgf opened
nRows =
     5
nCols =
     9
File YSDmodifiedByT.tgf closed

```

Compute tasks - data read relation

```

vertexLabels=horzcat(taskLabels,dataLabels',{'Task modifies Data'});
TmodifiesD=transposeR(DmodifiedByT)
isHomog=0
fileTmD=horzcat(tgfFilePre,'TmodifiesD','.tgf')
% Output graph data
tgfWrite(fileTmD,TreadsD,isHomog,vertexLabels,{});

```

```

TmodifiesD =
    0     0     0     0     0
    0     1     0     0     0
    0     1     0     0     0
    0     0     0     0     0
    0     0     1     0     0
    0     0     1     0     0
    0     0     0     0     0
    0     0     0     1     0
    0     0     0     1     0
isHomog =
    0
fileTmD =
YSTmodifiesD.tgf
File YSTmodifiesD.tgf opened
nRows =
    9
nCols =
    5
File YSTmodifiesD.tgf closed

```

Set up tool data and relations

Tools: CAD / g1 EngCalculations / g2 Spreadsheet / g3 WordProcessing / g4 Data - Tool(G) relations

```

toolLabels={'CAD','EngCalculations', ...
            'Spreadsheet','WordProcessing'}
% Data requires tool
% CAD / EngCalculations / Spreadsheet / WordProcessing
% Toolbugger
DrequiresG=([0 0 0 1 0 ; ... % ConceptPlan
             1 0 0 0 0; ... % ArchitecturalDrawings
             0 1 1 1 0; ... % StructuralDrawings
             1 0 0 0 0; ... % ConcreteDrawings
             0 0 0 0 0]) % Debugger
vertexLabels=horzcat(dataLabels,toolLabels',{'Data requires Tool'});
isHomog=0
fileDG=horzcat(tgfFilePre,'DrequiresG','.tgf')
tgfWrite(fileDG,DrequiresG,isHomog,vertexLabels,{});

```

```

toolLabels =
    'CAD'      'EngCalculations'      'Spreadsheet'      'WordProcessing'
DrequiresG =
    0     0     0     1     0
    1     0     0     0     0
    0     1     1     1     0
    1     0     0     0     0
    0     0     0     0     0
isHomog =
    0
fileDG =
YSDrequiresG.tgf
File YSDrequiresG.tgf opened
nRows =
    5
nCols =
    5
File YSDrequiresG.tgf closed

```

Person / Task / Data graph output

Output Person / Task / Data graphs - Data read only


```
makePersonTaskDataGraphs(tgfFilePre,personLabels,taskLabels,dataLabels, ...
    PexecutesT,TcreatesD,TreadsD,TmodifiesD,DrequiresG)
```

```
MATLAB implementation of relational algebra boolean matrix operations in inline functions
nPerson =
    5
nTask =
    9
nDataC =
    5
nDataR =
    5
nDataM =
    5
nDataset =
    5
nVertex =
    19
graphvertexLabels =
    Columns 1 through 7
    'Architect' {} {} {} {} [1x26 char] [1x26 char]
    Columns 8 through 15
    {} {} {} {} {} {} {} 'ConceptPlan'
    Columns 16 through 21
    [1x21 char] {} {} {} 'Tasks & Data for' 'Architect'
filename =
    YSArchitect.tgf
File YSArchitect.tgf opened
nRows =
    19
nCols =
    19
File YSArchitect.tgf closed
graphvertexLabels =
    Columns 1 through 11
    {} {} {} {} {} {} {} {} {} {} {}
    Columns 12 through 19
    {} {} {} {} {} {} {} {} {}
    Columns 20 through 21
    'Tasks & Data for' 'Client'
filename =
    YSClient.tgf
File YSClient.tgf opened
nRows =
    19
nCols =
    19
File YSClient.tgf closed
graphvertexLabels =
    Columns 1 through 8
    {} {} 'StructuralEngineer' {} {} {} {} {}
    Columns 9 through 15
    [1x23 char] [1x22 char] {} {} [1x20 char] {} {}
    Columns 16 through 19
    [1x21 char] 'StructuralDrawings' 'ConcreteDrawings' {}
    Columns 20 through 21
    'Tasks & Data for' 'StructuralEngineer'
filename =
    YSStructuralEngineer.tgf
File YSStructuralEngineer.tgf opened
nRows =
    19
nCols =
    19
File YSStructuralEngineer.tgf closed
graphvertexLabels =
    Columns 1 through 9
    {} {} {} 'Technologist' {} {} {} {} {}
```

```

Columns 10 through 16
    {}    {}    [1x21 char]    [1x20 char]    {}    {}    [1x21 char]
Columns 17 through 20
    'StructuralDrawings'    'ConcreteDrawings'    {}    'Tasks & Data for'
Column 21
    'Technologist'
filename =
YSTechnologist.tgf
File YSTechnologist.tgf opened
nRows =
    19
nCols =
    19
File YSTechnologist.tgf closed
graphvertexLabels =
Columns 1 through 7
    {}    {}    {}    {}    'CheckingEngineer'    {}    {}
Columns 8 through 13
    [1x24 char]    {}    {}    [1x21 char]    {}    {}
Columns 14 through 17
    'ConcreteDesignCheck'    {}    [1x21 char]    'StructuralDrawings'
Columns 18 through 21
    'ConcreteDrawings'    {}    'Tasks & Data for'    'CheckingEngineer'
filename =
YSCheckingEngineer.tgf
File YSCheckingEngineer.tgf opened
nRows =
    19
nCols =
    19
File YSCheckingEngineer.tgf closed

```

Set up status information

Status values: PreliminaryConcept / s1 DesignedEngineered / s2 Finalised / s3 Checked / s4

```

statusLabels={'PreliminaryConcept','DesignedEngineered', ...
    'Finalised','Checked'}
nStatus=size(statusLabels,2)
% 9 tasks by 4 status values for
% 5 dataSets for debug
TwritesDS{1}=[0 1 0 0 ; ...
    0 0 0 0 ; ...
    0 0 0 0 ; ...
    0 0 1 0 ; ...
    0 0 0 0 ; ...
    0 0 0 0 ; ...
    0 0 0 1 ; ...
    0 0 0 0 ; ...
    0 0 0 0 ]
TwritesDS{2}=[0 0 0 0 ; ...
    0 0 0 0 ; ...
    0 0 0 0 ; ...
    0 0 1 0 ; ...
    0 0 0 0 ; ...
    0 0 0 0 ; ...
    0 0 0 0 ; ...
    0 0 0 0 ; ...
    0 0 0 0 ]
TwritesDS{3}=[0 0 0 0 ; ...
    0 1 0 0 ; ...
    0 0 0 0 ; ...
    0 0 0 0 ; ...
    0 0 0 0 ; ...
    0 0 0 0 ; ...
    0 0 0 0 ; ...
    0 0 0 1 ; ...

```

```

        0 0 0 0 ]
TwritesDS{4}=[0 0 0 0 ; ...
        0 0 0 0 ; ...
        0 1 0 0 ; ...
        0 0 0 0 ; ...
        0 0 0 0 ; ...
        0 0 1 0 ; ...
        0 0 0 0 ; ...
        0 0 0 0 ; ...
        0 0 0 1]
TwritesDS{5}=[0 0 0 0 ; ...
        0 0 0 0 ; ...
        0 0 0 0 ; ...
        0 0 0 0 ; ...
        0 0 0 0 ; ...
        0 0 0 0 ; ...
        0 0 0 0 ; ...
        0 0 0 0]

```

```

statusLabels =
    'PreliminaryConcept'    'DesignedEngineered'    'Finalised'    'Checked'
nStatus =
    4
TwritesDS =
    [9x4 double]
TwritesDS =
    [9x4 double]    [9x4 double]
TwritesDS =
    [9x4 double]    [9x4 double]    [9x4 double]
TwritesDS =
    [9x4 double]    [9x4 double]    [9x4 double]    [9x4 double]
TwritesDS =
    Columns 1 through 4
    [9x4 double]    [9x4 double]    [9x4 double]    [9x4 double]
    Column 5
    [9x4 double]

```

Solution

Solution parameters

```

nPerson=size(PexecutesT,1)
nTask=size(PexecutesT,2)
nDataC=size(TcreatesD,2)
nDataR=size(TreadsD,2)
nDataM=size(TmodifiesD,2)
nDataset=max([nDataC,nDataR,nDataM])
%
PreadsD=productR(PexecutesT,TreadsD)
vertexLabels=horzcat(personLabels,dataLabels',{'Person reads Data with Status'});
isHomog=0
filePD=horzcat(tgfFilePre,'PreadsD','.tgf')
tgfWrite(filePD,PreadsD,isHomog,vertexLabels,{});
%
TrequiresG=productR(TreadsD,DrequiresG)
vertexLabels=horzcat(taskLabels,toolLabels',{'Task requires Tool with Status'});
isHomog=0
fileTG=horzcat(tgfFilePre,'TrequiresG','.tgf')
tgfWrite(fileTG,TrequiresG,isHomog,vertexLabels,{});
%
PrequiresG=productR(PreadsD,DrequiresG)
vertexLabels=horzcat(personLabels,toolLabels',{'Person requires Tool with Status'});
isHomog=0

```

```

filePG=horzcat(tgfFilePre,'RequiresG','.tgf')
tgfWrite(filePG,RequiresG,isHomog,vertexLabels,{});
% Set up basic Task-Task relation
TsequenceRule1T=productR(TcreatesD,unionR(DreadByT,DmodifiedByT))
vertexLabels=horzcat(taskLabels',{'Tasks sequence with Tasks - Rule 1'});
isHomog=1
fileTR1T=horzcat(tgfFilePre,'TsequenceRule1T','.tgf')
tgfWrite(fileTR1T,TsequenceRule1T,isHomog,vertexLabels,{});
% Testing
% TModStatus1T=productR(TwritesDS{1},DmodifiedByT)
% TModStatus2T=productR(TwritesDS{2},DmodifiedByT)
% TModStatus3T=productR(TwritesDS{3},DmodifiedByT)
% DModStatus1D=productR(transposeR(TwritesDS{1}),transposeR(DmodifiedByT))
% Rule 2 for task sequence
TsequenceRule2T=zeros(nTask,nTask);
% Determine data modified by task x task y combinations
for tx=1:nTask-1
    for ty=tx+1:nTask
        % extract modification datasets for tasks and combine with 'and'
        DmodifyTx=DmodifiedByT(:,tx);
        DmodifyTy=DmodifiedByT(:,ty);
        DmodifyTxAndTy=intersectionR(DmodifyTx,DmodifyTy);
        % display 'tx,ty,DmodTxAndTy',tx,ty,DmodifyTxAndTy
        % check status increase
        for id=1:nDataM
            if(DmodifyTxAndTy(id) == 1)
                rPx=0;
                rPy=0;
                for is=1:nStatus
                    if(TwritesDS{id}(tx,is))==1
                        rPx=is;
                    end
                    if(TwritesDS{id}(ty,is))==1
                        rPy=is;
                    end
                % display 'id,is,tx,ty,rPx,rPy',id,is,tx,ty,rPx,rPy
            end
            if(rPx<rPy) TsequenceRule2T(tx,ty)=1; end
        end
    end
end
end

```

```

nPerson =
    5
nTask =
    9
nDataC =
    5
nDataR =
    5
nDataM =
    5
nDataset =
    5
PreadsD =
    1    1    0    0    0
    0    0    0    0    0
    0    1    1    1    0
    0    1    1    1    0
    0    1    1    1    0
isHomog =
    0
filePD =
YSPreadsD.tgf
File YSPreadsD.tgf opened
nRows =
    5

```

```

nCols =
    5
File YSPreadsD.tgf closed
TrequiresG =
    0    0    0    1    0
    1    0    0    0    0
    1    0    0    0    0
    1    0    0    0    0
    0    1    1    1    0
    0    1    1    1    0
    1    1    1    1    0
    1    0    0    0    0
    1    0    0    0    0
isHomog =
    0
fileTG =
YSTrequiresG.tgf
File YSTrequiresG.tgf opened
nRows =
    9
nCols =
    5
File YSTrequiresG.tgf closed
PrequiresG =
    1    0    0    1    0
    0    0    0    0    0
    1    1    1    1    0
    1    1    1    1    0
    1    1    1    1    0
isHomog =
    0
filePG =
YSPrequiresG.tgf
File YSPrequiresG.tgf opened
nRows =
    5
nCols =
    5
File YSPrequiresG.tgf closed
TsequenceRule1T =
    0    1    1    1    0    0    1    0    0
    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0
    0    0    0    0    1    1    1    0    0
    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    1    1
    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0
isHomog =
    1
fileTR1T =
YSTsequenceRule1T.tgf
File YSTsequenceRule1T.tgf opened
nRows =
    9
nCols =
    9
File YSTsequenceRule1T.tgf closed

```

Output task sequence graphs

Rule 2 for task sequence

```

TsequenceRule2T
vertexLabels=horzcat(taskLabels,{'Tasks sequence with Tasks - Rule 2'});
isHomog=1

```

```

fileTR2T=horzcat(tgfFilePre,'TsequenceRule2T',' .tgf')
tgfWrite(fileTR2T,TsequenceRule2T,isHomog,vertexLabels,{});
TsequenceRule1Plus2T=unionR(TsequenceRule1T,TsequenceRule2T)
vertexLabels=horzcat(taskLabels,['Tasks sequence with Tasks - Rule 1&2']);
isHomog=1
fileTR12T=horzcat(tgfFilePre,'TsequenceRule1Plus2T',' .tgf')
tgfWrite(fileTR12T,TsequenceRule1Plus2T,isHomog,vertexLabels,{});
% Rule 3 for task sequence
TsequenceRule3T=zeros(nTask,nTask);
% Determine data modified by task x task y combinations
for tx=1:nTask-1
    for ty=tx+1:nTask
        % extract modification datasets for tasks and combine with 'and'
        DmodifyTx=DmodifiedByT(:,tx);
        DcreateTy=DcreatedByT(:,ty);
        DreadTy=DreadByT(:,ty);
        DmodifyTy=DmodifiedByT(:,ty);
        DmodifyTxAndReadTy=intersectionR(DmodifyTx,DreadTy);
        X=DmodifyTxAndReadTy;
        DcreateTyOrModifyTy=unionR(DcreateTy,DmodifyTy);
        Y=DcreateTyOrModifyTy;
        % display 'tx,ty,X,Y',tx,ty,X,Y
        % check status ranks
        for id=1:nDataM
            if(X (id) == 1)
                rminX=nStatus;
            for is=1:nStatus
                if(TwritesDS{id}(tx,is))==1
                    if(is<rminX) rminX=is ; end
                end
            end
            if(Y (id) == 1)
                rmaxY=0;
            for is=1:nStatus
                if(TwritesDS{id}(ty,is))==1
                    if(is>rmaxY) rmaxY=is ; end
                end
            end
            if(rminX<=rmaxY) TsequenceRule2T(tx,ty)=1; end
        end
    end
end
end % loop over ty
end % loop over tx
TsequenceRule3T

```

```

TsequenceRule2T =
    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    1
    0    0    0    0    0    0    0    0    0

isHomog =
    1
fileTR2T =
YSTsequenceRule2T.tgf
File YSTsequenceRule2T.tgf opened
nRows =
    9
nCols =
    9
File YSTsequenceRule2T.tgf closed
TsequenceRule1Plus2T =
    0    1    1    1    0    0    1    0    0

```

```

0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0
isHomog =
1
fileTR12T =
YSTsequenceRule1Plus2T.tgf
File YSTsequenceRule1Plus2T.tgf opened
nRows =
9
nCols =
9
File YSTsequenceRule1Plus2T.tgf closed
TsequenceRule3T =
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0

```

Combine task sequences and output task - step relation

```

TsequenceT=unionR(unionR(TsequenceRule1T,TsequenceRule2T),TsequenceRule3T)
vertexLabels=horzcat(taskLabels,{'Tasks sequence with Tasks - Rule 1&2&3'});
isHomog=1
fileTT=horzcat(tgffilePre,'TsequenceT','.tgf')
tgfWrite(fileTT,TsequenceT,isHomog,vertexLabels,{})
[Taskschedule]=TopolSortBFS(TsequenceT)
nSteps=size(Taskschedule,2)
stepLabel='Step';
for istep=1:nSteps
stepLabels{istep}=strcat(stepLabel,num2str(istep));
end
vertexLabels=horzcat(taskLabels,stepLabels,{'Tasks - Logical Steps'});
isHomog=0
fileTSchedule=horzcat(tgffilePre,'Taskschedule','.tgf')
tgfWrite(fileTSchedule,Taskschedule,isHomog,vertexLabels,{})

```

```

TsequenceT =
0 1 1 1 0 0 1 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0
isHomog =
1
fileTT =
YSTsequenceT.tgf
File YSTsequenceT.tgf opened
nRows =
9
nCols =

```

```

9
File YSTsequenceT.tgf closed
T =
  0   1   1   1   0   0   1   0   0
  0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0
  0   0   0   0   1   1   1   0   0
  0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   1   1
  0   0   0   0   0   0   0   0   1
  0   0   0   0   0   0   0   0   0
Topological Sort: Successor determination completed
level =
  2
Topological Sort: Successor determination completed
level =
  3
Topological Sort: Successor determination completed
level =
  4
Topological Sort: Successor determination completed
level =
  5
Topological Sort: Successor determination completed
level =
  6
Topological Sort: Logical taskschedule computed
TaskSchedule =
  1   0   0   0   0
  0   1   0   0   0
  0   1   0   0   0
  0   1   0   0   0
  0   0   1   0   0
  0   0   1   0   0
  0   0   1   0   0
  0   0   0   1   0
  0   0   0   0   1
Taskschedule =
  1   0   0   0   0
  0   1   0   0   0
  0   1   0   0   0
  0   1   0   0   0
  0   0   1   0   0
  0   0   1   0   0
  0   0   1   0   0
  0   0   0   1   0
  0   0   0   0   1
nSteps =
  5
isHomog =
  0
fileTSchedule =
YSTaskschedule.tgf
File YSTaskschedule.tgf opened
nRows =
  9
nCols =
  5
File YSTaskschedule.tgf closed

```


Appendix H

Process Model: Person-Task and Person-Data Graphs

H.1 Engineering Process Model Graphical Output Example - MATLAB Code

Contents

- Relation algebra MATLAB functionality
- Person-task relation and labels
- Task-persons relation
- Task-data relation
- Data - Tool(G) relations
- Process data to determine task and data per person

```
%.....  
% Project Model for typical Building Project  
% Person / task / data graphs  
%.....  
clc  
clear all  
format compact
```

Relation algebra MATLAB functionality

```
RelationalAlgebraBoolean  
tgfFilePre='YProcessPTD'
```

```
Error using ==> evalin  
Undefined function or variable 'RelationalAlgebraBoolean'.
```

Person-task relation and labels

Define persons executes tasks PT Tasks: Conceptualise / Plan / Design / Specify & Document / Take off Quantities / Build & Construct

```
PexecutesT=([1 0 0 0 0 0 ; ... % Client Owner  
            1 1 0 0 0 0 ; ... % Architect  
            0 0 1 1 0 0 ; ... % Engineer  
            0 0 0 1 1 0 ; ... % Quantity Surveyor  
            0 0 0 0 0 1 ]) % Constructor/Contractor  
% TgfWrite(tgfFile,R,vertexLabels,edgeLabels)  
personLabels={'ClientOwner','Architect','Engineer', ...  
             'QuantitySurveyor','ContractorBuilder'}
```

```

taskLabels={'Conceptualise','Plan','Eng Design', ...
            'Specify & Document', ...
            'Take off Quantities','Build & Construct'}
% add comment & heading strings at end of labels
vertexLabels=horzcat(personLabels,taskLabels',{'Person executes Task'})
isHomog=0
filePT=horzcat(tgfFilePre,'PexecutesT','.tgf')
tgfWrite(filePT,PexecutesT,isHomog,vertexLabels,{})

```

Task-persons relation

Compute tasks - persons TP

```

vertexLabels=horzcat(taskLabels,personLabels',{'Task executed by Person'})
TexecutedByP=transposeR(PexecutesT)
isHomog=0
fileTP=horzcat(tgfFilePre,'TexecutedByP','.tgf')
tgfWrite(fileTP,TexecutedByP,isHomog,vertexLabels,{})

```

Task-data relation

Task creates / modifies / reads data Concept plan = dwgs +document / Architects dwgs / Design calcs / Eng Plans / Specifications / Bill of Quantities / Construction pgms/sched / As built plans

```

dataLabels={'Concept plan','Architects dwgs','Design calculations', ...
            'Engineering dwgs','Specifications','Bill of Quantities', ...
            'Construction pgms/sched','As built dwgs'}
TcreatesD=([1 0 0 0 0 0 0 0 ; ... % Conceptualise
            0 1 0 0 0 0 0 0; ... % Plan
            0 0 1 1 0 0 0 0; ... % Engineering design
            0 0 0 0 1 0 0 0; ... % Specify and document
            0 0 0 0 0 1 0 0; ... % Take off quantities
            0 0 0 0 0 0 1 1]) % Construct
vertexLabels=horzcat(taskLabels,dataLabels',{'Task creates Data'})
isHomog=0
fileTcD=horzcat(tgfFilePre,'TcreatesD','.tgf')
tgfWrite(fileTcD,TcreatesD,isHomog,vertexLabels,{})
TreadsD= ([0 0 0 0 0 0 0 0 ; ... % Conceptualise
            1 0 0 0 0 0 0 0; ... % Plan
            1 1 0 0 0 0 0 0; ... % Engineering design
            0 1 1 0 0 0 0 0; ... % Specify and document
            0 1 0 1 1 0 0 0; ... % Take off quantities
            0 1 0 1 1 1 1 0]) % Construct
vertexLabels=horzcat(taskLabels,dataLabels',{'Task reads Data'})
isHomog=0
fileTrD=horzcat(tgfFilePre,'TreadsD','.tgf')
tgfWrite(fileTrD,TreadsD,isHomog,vertexLabels,{})
TmodifiesD= ([1 1 0 0 0 0 0 0; ... % Conceptualise
            0 1 0 0 0 0 0 0; ... % Plan
            0 0 1 1 0 0 0 0; ... % Engineering design
            0 0 0 0 1 0 0 0; ... % Specify and document
            0 0 0 0 0 1 0 0; ... % Take off quantities
            0 0 0 0 0 0 1 1]) % Construct
vertexLabels=horzcat(taskLabels,dataLabels',{'Task modifies Data'})
isHomog=0
fileTmD=horzcat(tgfFilePre,'TmodifiesD','.tgf')
tgfWrite(fileTmD,TmodifiesD,isHomog,vertexLabels,{})

```

Data - Tool(G) relations

Tools: Text processor / CAD / Eng Design SW / Quantities SW / Construction Planning SW

```

toolLabels={'Text processor','CAD','Eng Design SW', ...
            'Quantities SW','Construction Planning SW'}
% Data requires tool
DrequiresG=([1 1 0 0 0 ; ... % Conceptual plan requires txt processor & CAD
            0 1 0 0 0 ; ... % Arch plan requires CAD processor
            0 0 1 0 0 ; ... % Design calcs require Eng Design SW
            0 1 0 0 0 ; ... % Engineering plans require CAD
            1 0 0 0 0 ; ... % Specifications require txt processor
            0 0 0 1 0 ; ... % Bill of Quantities requires off Quant software
            0 0 0 0 1 ; ... % Construction plans & schedules Constr SW
            0 1 0 0 0]) % As built plans require CAD
vertexLabels=horzcat(dataLabels,toolLabels',{'Data requires Tool'})
isHomog=0
fileDG=horzcat(tgfFilePre,'DrequiresG','.tgf')
tgfWrite(fileDG,DrequiresG,isHomog,vertexLabels,{})

```

Process data to determine task and data per person

Output Person / Task / Data graphs

```

makePersonTaskDataGraphs(tgfFilePre,personLabels,taskLabels,dataLabels, ...
                        PexecutesT,TcreatesD,TreadsD,TmodifiesD,DrequiresG)

```

H.2 Process model graphical data output MATLAB function

The MATLAB code for the function reference is listed below.

```

function [] = makePersonTaskDataGraphs(tgfFilePre,personLabels,taskLabels,dataLabels, ...
                        PexecutesT,TcreatesD,TreadsD,TmodifiesD,DrequiresG)
% .....
% subroutine to make person task datagraphs
% tgfFilePre - tgfFilename first characters - ID of problem
% personLabels,taskLabels,dataLabels - Cell arrays with labels
% PexecutesT - Define persons executes tasks PT
% TcreatesD - Task creates data relation
% TreadsD - Task reads data relation
% TmodifiesD - Task modifies data relation
% DrequiresG - Data - Tool(G) relations
% .....
RelationalAlgebraBoolean
% Generate Person - Task - Data graphs looping over persons
% number of persons in boolean matrix
nPerson=size(PexecutesT,1)
nTask=size(PexecutesT,2)
nDataC=size(TcreatesD,2)
nDataR=size(TreadsD,2)
nDataM=size(TmodifiesD,2)
nDataset=max([nDataC,nDataR,nDataM])
nVertex=nPerson+nTask+nDataset
% set up boolean matrix - person & task & data vertices
for iPerson=1:nPerson
    PersonTaskDataMatrix=zeros(nVertex,nVertex);
% prepare vertex labels - only non blank entries output
for iLabel=1:nPerson+nTask+nDataset
    vertexLabels{iLabel}={};
end
% extract row to get tasks per person
tasks=PexecutesT(iPerson,:);
for iTask=1:nTask
% add person - task link into graph
    iRow=iPerson;
    iCol=nPerson+iTask;
    if tasks(iTask)==1
        PersonTaskDataMatrix(iCol,iRow)=tasks(iTask);
        vertexLabels{iRow}=personLabels{iPerson};
        vertexLabels{iCol}=taskLabels{iTask};
    end
end

```

```

% set up task - data links
if(nDataC > 0) dataCreate=TcreatesD(iTask,:) ; end
if(nDataR > 0) dataRead=TreadsD(iTask,:) ; end
if(nDataM > 0) dataModify=TmodifiesD(iTask,:); end
for iData=1:nDataset
    % add data entry to graph
    iRow=nPerson+iTask;
    iCol=nPerson+nTask+iData;
    if (nDataC > 0)
        if (dataCreate(iData)==1)
            PersonTaskDataMatrix(iRow,iCol)=dataCreate(iData);
            vertexLabels{iCol}=dataLabels{iData};
        end;end
    if (nDataR > 0)
        if (dataRead(iData)==1)
            PersonTaskDataMatrix(iCol,iRow)=dataRead(iData);
            vertexLabels{iCol}=dataLabels{iData};
        end;end
    if (nDataM > 0)
        if (dataModify(iData)==1)
            PersonTaskDataMatrix(iCol,iRow)=dataModify(iData);
            PersonTaskDataMatrix(iRow,iCol)=dataModify(iData);
            vertexLabels{iCol}=dataLabels{iData};
        end;end
    end
end
end
graphvertexLabels=horzcat(vertexLabels,horzcat(cellstr('Tasks & Data for ...
'),cellstr(personLabels{iPerson})))
isHomog=1;
filename=horzcat(tgfFilePre, personLabels{iPerson},'.tgf')
tgfWrite(filename, PersonTaskDataMatrix, isHomog, graphvertexLabels, {})
end

```

Appendix I

Process Model: Task sequence using data status

I.1 Engineering Process Model Example with data status - MATLAB Code

Contents

- Relational algebra MATLAB functionality
- Persons, Tasks and Person - task relations
- Datasets and Data-task relations
- Data-Tool relations
- Output data for graphical display by yEd
- Data status values and Task Data status relations
- Set up and compute solution
- Task-Task relation by Rule 1
- Task-Task relation by Rule 2
- Task-Task relation by Rule 3
- Combine Rules 1, 2 and 3 and determine task logical sequence

```
%.....  
% Process Model for typical Design Project - Example from Tutorial  
% With status settings  
% 2006/04/16  
%.....  
clc  
clear all  
format compact
```

Relational algebra MATLAB functionality

```
RelationalAlgebraBoolean  
tgfFilePre='YS'
```

```
MATLAB implementation of relational algebra boolean matrix operations in inline functions  
tgfFilePre =  
YS
```

Persons, Tasks and Person - task relations

```
personLabels={'Architect','Client','StructuralEngineer', ...  
             'Technologist','CheckingEngineer'}  
taskLabels= {'ArchitecturalLayoutConcept', ...  
            'ArchitecturalDesignDetail ', ...  
            'ArchitecturalDesignCheck', ...  
            'StructuralLayoutConcept', ...  
            'StructuralDesignDetail', ...
```

```

        'StructuralDesignCheck', ...
        'ConcreteLayoutConcept', ...
        'ConcreteDesignDetail', ...
        'ConcreteDesignCheck'}
% Define persons executes tasks PT
% Tasks: ArchitecturalLayoutConcept / t1
%       ArchitecturalDesignDetail / t2
%       ArchitecturalDesignCheck / t3
%       StructuralLayoutConcept / t4
%       StructuralDesignDetail / t5
%       StructuralDesignCheck / t6
%       ConcreteLayoutConcept / t7
%       ConcreteDesignDetail / t8
%       ConcreteDesignCheck / t9
PexecutesT=([1 1 0 0 0 0 0 0 0 ; ... % Architect / p1
            0 0 0 0 0 0 0 0 0 ; ... % Client / p2
            0 0 0 1 1 0 0 1 0 ; ... % StructuralEngineer /p3
            0 0 0 0 0 0 1 1 0 ; ... % Technologist /p4
            0 0 1 0 0 1 0 0 1 ]) % CheckingEngineer /p5
% add comment & heading strings at end of labels
vertexLabels=horzcat(personLabels,taskLabels,{'Person executes Task'});
isHomog=0
filePT=horzcat(tgfFilePre,'PexecutesT','.tgf')
tgfWrite(filePT,PexecutesT,isHomog,vertexLabels,{});
% Compute tasks - persons TP
vertexLabels=horzcat(taskLabels,personLabels,{'Task executed by Person'});
TexecutedByP=transposeR(PexecutesT)
isHomog=0
fileTP=horzcat(tgfFilePre,'TexecutedByP','.tgf')
tgfWrite(fileTP,TexecutedByP,isHomog,vertexLabels,{});

```

```

personLabels =
Columns 1 through 3
    'Architect'    'Client'    'StructuralEngineer'
Columns 4 through 5
    'Technologist'    'CheckingEngineer'
taskLabels =
Columns 1 through 4
    [1x26 char]    [1x26 char]    [1x24 char]    [1x23 char]
Columns 5 through 8
    [1x22 char]    [1x21 char]    [1x21 char]    [1x20 char]
Column 9
    'ConcreteDesignCheck'
PexecutesT =
    1     1     0     0     0     0     0     0     0
    0     0     0     0     0     0     0     0     0
    0     0     0     1     1     0     0     1     0
    0     0     0     0     0     0     1     1     0
    0     0     1     0     0     1     0     0     1
isHomog =
    0
filePT =
YSPexecutesT.tgf
File YSPexecutesT.tgf opened
nRows =
    5
nCols =
    9
File YSPexecutesT.tgf closed
TexecutedByP =
    1     0     0     0     0
    1     0     0     0     0
    0     0     0     0     1
    0     0     1     0     0
    0     0     1     0     0
    0     0     0     0     1
    0     0     0     1     0
    0     0     1     1     0

```

```

    0    0    0    0    1
isHomog =
    0
fileTP =
YSTexecutedByP.tgf
File YSTexecutedByP.tgf opened
nRows =
    9
nCols =
    5
File YSTexecutedByP.tgf closed

```

Datasets and Data-task relations

Datasets ConceptPlan/ d1 ArchitecturalDrawings/ d2 StructuralDrawings/ d3 ConcreteDrawings /d4

```

dataLabels={'ConceptPlan', ...
            'ArchitecturalDrawings', ...
            'StructuralDrawings', ...
            'ConcreteDrawings'}
% Data is created by Task
DcreatedByT=[0 0 0 0 0 0 0 0; ... % ConceptPlan
             1 0 0 0 0 0 0 0; ... % ArchitecturalDrawings
             0 0 0 1 0 0 0 0; ... % StructuralDrawings
             0 0 0 0 0 0 1 0] % ConcreteDrawings
% output graph data
vertexLabels=horzcat(dataLabels,taskLabels',{'Data created by Task'});
isHomog=0
fileDcT=horzcat(tgfFilePre,'DcreatedByT','.tgf')
tgfWrite(fileDcT,DcreatedByT,isHomog,vertexLabels,{});
% Compute tasks - data create relation
vertexLabels=horzcat(taskLabels,dataLabels',{'Task creates Data'});
TcreatesD=transposeR(DcreatedByT)
isHomog=0
fileTcD=horzcat(tgfFilePre,'TcreatesD','.tgf')
tgfWrite(fileTcD,TcreatesD,isHomog,vertexLabels,{});
% Data is read by Task
DreadByT= ([ 1 0 0 0 0 0 0 0; ... % ConceptPlan
            0 1 1 1 0 0 1 0; ... % ArchitecturalDrawings
            0 0 0 0 1 1 1 0; ... % StructuralDrawings
            0 0 0 0 0 0 0 1]) % ConcreteDrawings
% output graph data
vertexLabels=horzcat(dataLabels,taskLabels',{'Data read by Task'});
isHomog=0
fileDrT=horzcat(tgfFilePre,'DreadByT','.tgf')
tgfWrite(fileDrT,DreadByT,isHomog,vertexLabels,{});
% Compute tasks - data read relation
vertexLabels=horzcat(taskLabels,dataLabels',{'Task reads Data'});
TreadsD=transposeR(DreadByT)
isHomog=0
fileTrD=horzcat(tgfFilePre,'TreadsD','.tgf')
tgfWrite(fileTrD,TreadsD,isHomog,vertexLabels,{});
% Data is modified by Task
DmodifiedByT= ([ 0 0 0 0 0 0 0 0; ... % ConceptPlan
                0 1 1 0 0 0 0 0; ... % ArchitecturalDrawings
                0 0 0 0 1 1 0 0; ... % StructuralDrawings
                0 0 0 0 0 0 0 1]) % ConcreteDrawings
% output graph data
vertexLabels=horzcat(dataLabels,taskLabels',{'Data modified by Task'});
isHomog=0
fileDmT=horzcat(tgfFilePre,'DmodifiedByT','.tgf')
tgfWrite(fileDmT,DmodifiedByT,isHomog,vertexLabels,{});
% Compute tasks - data read relation
vertexLabels=horzcat(taskLabels,dataLabels',{'Task modifies Data'});
TmodifiesD=transposeR(DmodifiedByT)
isHomog=0
fileTmD=horzcat(tgfFilePre,'TmodifiesD','.tgf')
tgfWrite(fileTmD,TmodifiesD,isHomog,vertexLabels,{});

```

```

dataLabels =
  Columns 1 through 3
    'ConceptPlan' [1x21 char]    'StructuralDrawings'
  Column 4
    'ConcreteDrawings'
DcreatedByT =
  0  0  0  0  0  0  0  0  0
  1  0  0  0  0  0  0  0  0
  0  0  0  1  0  0  0  0  0
  0  0  0  0  0  0  1  0  0
isHomog =
  0
fileDcT =
YSDcreatedByT.tgf
File YSDcreatedByT.tgf opened
nRows =
  4
nCols =
  9
File YSDcreatedByT.tgf closed
TcreatesD =
  0  1  0  0
  0  0  0  0
  0  0  0  0
  0  0  1  0
  0  0  0  0
  0  0  0  0
  0  0  0  1
  0  0  0  0
  0  0  0  0
isHomog =
  0
fileTcD =
YSTcreatesD.tgf
File YSTcreatesD.tgf opened
nRows =
  9
nCols =
  4
File YSTcreatesD.tgf closed
DreadByT =
  1  0  0  0  0  0  0  0  0
  0  1  1  1  0  0  1  0  0
  0  0  0  0  0  1  1  0  0
  0  0  0  0  0  0  0  1  1
isHomog =
  0
fileDrT =
YSDreadByT.tgf
File YSDreadByT.tgf opened
nRows =
  4
nCols =
  9
File YSDreadByT.tgf closed
TreadsD =
  1  0  0  0
  0  1  0  0
  0  1  0  0
  0  1  0  0
  0  0  1  0
  0  0  1  0
  0  1  1  0
  0  0  0  1
  0  0  0  1
isHomog =
  0
fileTrD =
YSTreadsD.tgf
File YSTreadsD.tgf opened

```



```

nRows =
    9
nCols =
    4
File YSTreadsD.tgf closed
DmodifiedByT =
    0    0    0    0    0    0    0    0    0
    0    1    1    0    0    0    0    0    0
    0    0    0    0    0    1    1    0    0
    0    0    0    0    0    0    0    1    1
isHomog =
    0
fileDmT =
YSDmodifiedByT.tgf
File YSDmodifiedByT.tgf opened
nRows =
    4
nCols =
    9
File YSDmodifiedByT.tgf closed
TmodifiesD =
    0    0    0    0
    0    1    0    0
    0    1    0    0
    0    0    0    0
    0    0    1    0
    0    0    1    0
    0    0    0    0
    0    0    0    1
    0    0    0    1
isHomog =
    0
fileTmD =
YSTmodifiesD.tgf
File YSTmodifiesD.tgf opened
nRows =
    9
nCols =
    4
File YSTmodifiesD.tgf closed

```

Data-Tool relations

Tools: CAD / g1 EngCalculations / g2 Spreadsheet / g3 WordProcessing / g4 Data - Tool(G) relations

```

toolLabels={'CAD','EngCalculations', ...
            'Spreadsheet','WordProcessor'}
% Data requires tool
% CAD / EngCalculations / Spreadsheet / WordProcessing
% Toolbugger
DrequiresG=([0 0 0 1 0 ; ... % ConceptPlan
             1 0 0 0 0; ... % ArchitecturalDrawings
             0 1 1 1 0; ... % StructuralDrawings
             1 0 0 0 0]) ... % ConcreteDrawings
%             0 0 0 0 0]) % Debugger
vertexLabels=horzcat(dataLabels,toolLabels',{'Data requires Tool'});
isHomog=0
fileDG=horzcat(tgfFilePre,'DrequiresG','.tgf')
tgfWrite(fileDG,DrequiresG,isHomog,vertexLabels,{});

```

```

toolLabels =
    Columns 1 through 3
    'CAD'      'EngCalculations'  'Spreadsheet'
    Column 4

```

```
'WordProcessor'
DrequiresG =
  0    0    0    1    0
  1    0    0    0    0
  0    1    1    1    0
  1    0    0    0    0
isHomog =
  0
fileDG =
YSDrequiresG.tgf
File YSDrequiresG.tgf opened
nRows =
  4
nCols =
  5
File YSDrequiresG.tgf closed
```

Output data for graphical display by yEd

Output Person / Task / Data graphs - Data read only

```
makePersonTaskDataGraphs(tgfFilePre,personLabels,taskLabels,dataLabels, ...
                          PexecutesT,TcreatesD,TreadsD,TmodifiesD,DrequiresG)

%
% Set up status information
```

```

MATLAB implementation of relational algebra boolean matrix operations in inline functions
nPerson =
    5
nTask =
    9
nDataC =
    4
nDataR =
    4
nDataM =
    4
nDataset =
    4
nVertex =
    18
graphvertexLabels =
    Columns 1 through 6
        'Architect'    {}    {}    {}    [1x26 char]
    Columns 7 through 13
        [1x26 char]    {}    {}    {}    {}    {}
    Columns 14 through 18
        {}    'ConceptPlan'    [1x21 char]    {}    {}
    Columns 19 through 20
        'Tasks & Data for'    'Architect'
filename =
YSArchitect.tgf
File YSArchitect.tgf opened
nRows =
    18
nCols =
    18
File YSArchitect.tgf closed
graphvertexLabels =
    Columns 1 through 8
        {}    {}    {}    {}    {}    {}    {}    {}
    Columns 9 through 16
        {}    {}    {}    {}    {}    {}    {}    {}
    Columns 17 through 20

```

```

    {}    {}    'Tasks & Data for'    'Client'
filename =
YSClient.tgf
File YSClient.tgf opened
nRows =
    18
nCols =
    18
File YSClient.tgf closed
graphvertexLabels =
    Columns 1 through 6
        {}    {}    'StructuralEngineer'    {}    {}    {}
    Columns 7 through 12
        {}    {}    [1x23 char]    [1x22 char]    {}    {}
    Columns 13 through 16
        [1x20 char]    {}    {}    [1x21 char]
    Columns 17 through 18
        'StructuralDrawings'    'ConcreteDrawings'
    Columns 19 through 20
        'Tasks & Data for'    'StructuralEngineer'
filename =
YSStructuralEngineer.tgf
File YSStructuralEngineer.tgf opened
nRows =
    18
nCols =
    18
File YSStructuralEngineer.tgf closed
graphvertexLabels =
    Columns 1 through 7
        {}    {}    {}    'Technologist'    {}    {}    {}
    Columns 8 through 13
        {}    {}    {}    {}    [1x21 char]    [1x20 char]
    Columns 14 through 17
        {}    {}    [1x21 char]    'StructuralDrawings'
    Columns 18 through 19
        'ConcreteDrawings'    'Tasks & Data for'
    Column 20
        'Technologist'
filename =
YSTechnologist.tgf
File YSTechnologist.tgf opened
nRows =
    18
nCols =
    18
File YSTechnologist.tgf closed
graphvertexLabels =
    Columns 1 through 6
        {}    {}    {}    {}    'CheckingEngineer'    {}
    Columns 7 through 12
        {}    [1x24 char]    {}    {}    [1x21 char]    {}
    Columns 13 through 16
        {}    'ConcreteDesignCheck'    {}    [1x21 char]
    Columns 17 through 18
        'StructuralDrawings'    'ConcreteDrawings'
    Columns 19 through 20
        'Tasks & Data for'    'CheckingEngineer'
filename =
YSCheckingEngineer.tgf
File YSCheckingEngineer.tgf opened
nRows =
    18
nCols =
    18
File YSCheckingEngineer.tgf closed

```

Data status values and Task Data status relations

Status values: PreliminaryConcept / s1 DesignedEngineered / s2 Finalised / s3 Checked / s4

```

statusLabels={'Preliminary_Concept','DesignedEngineered', ...
              'Finalised','Checked'}
nStatus=size(statusLabels,2)
% 9 tasks by 4 status values for
% 5 dataSets for debug
TwritesDS{1}=[0 1 0 0 ; ...
              0 0 0 0 ; ...
              0 0 0 0 ; ...
              0 0 1 0 ; ...
              0 0 0 0 ; ...
              0 0 0 0 ; ...
              0 0 0 1 ; ...
              0 0 0 0 ; ...
              0 0 0 0 ]
TwritesDS{2}=[0 0 0 0 ; ...
              0 0 0 0 ; ...
              0 0 0 0 ; ...
              0 0 1 0 ; ...
              0 0 0 0 ; ...
              0 0 0 0 ; ...
              0 0 0 0 ; ...
              0 0 0 0 ; ...
              0 0 0 0 ]
TwritesDS{3}=[0 0 0 0 ; ...
              0 1 0 0 ; ...
              0 0 0 0 ; ...
              0 0 0 0 ; ...
              0 0 0 0 ; ...
              0 0 0 0 ; ...
              0 0 0 0 ; ...
              0 0 0 1 ; ...
              0 0 0 0 ]
TwritesDS{4}=[0 0 0 0 ; ...
              0 0 0 0 ; ...
              0 1 0 0 ; ...
              0 0 0 0 ; ...
              0 0 0 0 ; ...
              0 0 1 0 ; ...
              0 0 0 0 ; ...
              0 0 0 0 ; ...
              0 0 0 1]

```

```

statusLabels =
  Columns 1 through 2
    'Preliminary_Concept'    'DesignedEngineered'
  Columns 3 through 4
    'Finalised'    'Checked'
nStatus =
    4
TwritesDS =
    [9x4 double]
TwritesDS =
    [9x4 double]    [9x4 double]
TwritesDS =
    [9x4 double]    [9x4 double]    [9x4 double]
TwritesDS =
  Columns 1 through 3
    [9x4 double]    [9x4 double]    [9x4 double]
  Column 4
    [9x4 double]

```

Set up and compute solution

Solution parameters

```

nPerson=size(PexecutesT,1)
nTask=size(PexecutesT,2)
nDataC=size(TcreatesD,2)
nDataR=size(TreadsD,2)
nDataM=size(TmodifiesD,2)
nDataset=max([nDataC,nDataR,nDataM])
%
PreadsD=productR(PexecutesT,TreadsD)
vertexLabels=horzcat(personLabels,dataLabels',{'Person reads Data with Status'});
isHomog=0
filePD=horzcat(tgfFilePre,'PreadsD',' .tgf')
tgfWrite(filePD,PreadsD,isHomog,vertexLabels,{});
%
TrequiresG=productR(TreadsD,DrequiresG)
vertexLabels=horzcat(taskLabels,toolLabels {'Task requires Tool with Status'});
isHomog=0
fileTG=horzcat(tgfFilePre,'TrequiresG',' .tgf')
tgfWrite(fileTG,TrequiresG,isHomog,vertexLabels,{});
%
PrequiresG=productR(PreadsD,DrequiresG)
vertexLabels=horzcat(personLabels,toolLabels {'Person requires Tool with Status'});
isHomog=0
filePG=horzcat(tgfFilePre,'PrequiresG',' .tgf')
tgfWrite(filePG,PrequiresG,isHomog,vertexLabels,{});

```

```

nPerson =
    5
nTask =
    9
nDataC =
    4
nDataR =
    4
nDataM =
    4
nDataset =
    4
PreadsD =
    1    1    0    0
    0    0    0    0
    0    1    1    1
    0    1    1    1
    0    1    1    1
isHomog =
    0
filePD =
YSPreadsD.tgf
File YSPreadsD.tgf opened
nRows =
    5
nCols =
    4
File YSPreadsD.tgf closed
TrequiresG =
    0    0    0    1    0
    1    0    0    0    0
    1    0    0    0    0
    1    0    0    0    0
    0    1    1    1    0
    0    1    1    1    0
    1    1    1    1    0
    1    0    0    0    0
    1    0    0    0    0
isHomog =
    0
fileTG =
YSTrequiresG.tgf
File YSTrequiresG.tgf opened

```

```

nRows =
    9
nCols =
    5
File YSTrequiresG.tgf closed
PrequiresG =
    1    0    0    1    0
    0    0    0    0    0
    1    1    1    1    0
    1    1    1    1    0
    1    1    1    1    0
isHomog =
    0
filePG =
YSTPrequiresG.tgf
File YSPrequiresG.tgf opened
nRows =
    5
nCols =
    5
File YSPrequiresG.tgf closed

```

Task-Task relation by Rule 1

set up basic Task-Task relation - rule 1

```

TsequenceRule1T=productR(TcreatesD,unionR(DreadByT,DmodifiedByT))
vertexLabels=horzcat(taskLabels',{'Tasks sequence with Tasks - Rule 1'});
isHomog=1
fileTR1T=horzcat(tgfFilePre,'TsequenceRule1T','.tgf')
tgfWrite(fileTR1T,TsequenceRule1T,isHomog,vertexLabels,{});

```

```

TsequenceRule1T =
    0    1    1    1    0    0    1    0    0
    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0
    0    0    0    0    1    1    1    0    0
    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    1    1
    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0
isHomog =
    1
fileTR1T =
YSTsequenceRule1T.tgf
File YSTsequenceRule1T.tgf opened
nRows =
    9
nCols =
    9
File YSTsequenceRule1T.tgf closed

```

Task-Task relation by Rule 2

Rule 2 for task sequence

```

TsequenceRule2T=zeros(nTask,nTask);
% Determine data modified by task x task y combinations
for tx=1:nTask-1

```

```

    for ty=tx+1:nTask
% extract modification datasets for tasks and combine with 'and'
DmodifyTx=DmodifiedByT(:,tx);
DmodifyTy=DmodifiedByT(:,ty);
DmodifyTxAndTy=intersectionR(DmodifyTx,DmodifyTy);
display 'tx,ty,DmodTxAndTy',tx,ty,DmodifyTxAndTy
% check status increase
for id=1:nDataM
    %
    if(DmodifyTxAndTy(id) == 1)
rPx=0;
rPy=0;
for is=1:nStatus
    if(TwritesDS{id}(tx,is))==1
        rPx=is;
    end
    if(TwritesDS{id}(ty,is))==1
        rPy=is;
    end
    display 'id,is,tx,ty,rPx,rPy',id,is,tx,ty,rPx,rPy
end
    if(rPx<rPy) TsequenceRule2T(tx,ty)=1; end
end
end
end
TsequenceRule2T
vertexLabels=horzcat(taskLabels',{'Tasks sequence with Tasks - Rule 2'});
isHomog=1
fileTR2T=horzcat(tgfFilePre,'TsequenceRule2T',' .tgf')
tgfWrite(fileTR2T,TsequenceRule2T,isHomog,vertexLabels,{});
TsequenceRule1Plus2T=unionR(TsequenceRule1T,TsequenceRule2T)
vertexLabels=horzcat(taskLabels',{'Tasks sequence with Tasks - Rule 1&2'});
isHomog=1
fileTR12T=horzcat(tgfFilePre,'TsequenceRule1Plus2T',' .tgf')
tgfWrite(fileTR12T,TsequenceRule1Plus2T,isHomog,vertexLabels,{});

```

```

tx,ty,DmodTxAndTy
tx =
    1
ty =
    2
DmodifyTxAndTy =
    0
    0
    0
    0
tx,ty,DmodTxAndTy
tx =
    1
ty =
    3
DmodifyTxAndTy =
    0
    0
    0
    0
tx,ty,DmodTxAndTy
tx =
    1
ty =
    4
DmodifyTxAndTy =
    0
    0
    0
    0
tx,ty,DmodTxAndTy

```

```

tx =
  1
ty =
  5
DmodifyTxAndTy =
  0
  0
  0
  0
tx,ty,DmodTxAndTy
tx =
  1
ty =
  6
DmodifyTxAndTy =
  0
  0
  0
  0
tx,ty,DmodTxAndTy
tx =
  1
ty =
  7
DmodifyTxAndTy =
  0
  0
  0
  0
tx,ty,DmodTxAndTy
tx =
  1
ty =
  8
DmodifyTxAndTy =
  0
  0
  0
  0
tx,ty,DmodTxAndTy
tx =
  1
ty =
  9
DmodifyTxAndTy =
  0
  0
  0
  0
tx,ty,DmodTxAndTy
tx =
  2
ty =
  3
DmodifyTxAndTy =
  0
  1
  0
  0
id,is,tx,ty,rPx,rPy
id =
  2
is =
  1
tx =
  2
ty =
  3
rPx =
  0
rPy =

```



```

    0
id,is,tx,ty,rPx,rPy
id =
    2
is =
    2
tx =
    2
ty =
    3
rPx =
    0
rPy =
    0
id,is,tx,ty,rPx,rPy
id =
    2
is =
    3
tx =
    2
ty =
    3
rPx =
    0
rPy =
    0
id,is,tx,ty,rPx,rPy
id =
    2
is =
    4
tx =
    2
ty =
    3
rPx =
    0
rPy =
    0
tx,ty,DmodTxAndTy
tx =
    2
ty =
    4
DmodifyTxAndTy =
    0
    0
    0
    0
tx,ty,DmodTxAndTy
tx =
    2
ty =
    5
DmodifyTxAndTy =
    0
    0
    0
    0
tx,ty,DmodTxAndTy
tx =
    2
ty =
    6
DmodifyTxAndTy =
    0
    0
    0
    0
tx,ty,DmodTxAndTy

```

```

tx =
    2
ty =
    7
DmodifyTxAndTy =
    0
    0
    0
    0
tx,ty,DmodTxAndTy
tx =
    2
ty =
    8
DmodifyTxAndTy =
    0
    0
    0
    0
tx,ty,DmodTxAndTy
tx =
    2
ty =
    9
DmodifyTxAndTy =
    0
    0
    0
    0
tx,ty,DmodTxAndTy
tx =
    3
ty =
    4
DmodifyTxAndTy =
    0
    0
    0
    0
tx,ty,DmodTxAndTy
tx =
    3
ty =
    5
DmodifyTxAndTy =
    0
    0
    0
    0
tx,ty,DmodTxAndTy
tx =
    3
ty =
    6
DmodifyTxAndTy =
    0
    0
    0
    0
tx,ty,DmodTxAndTy
tx =
    3
ty =
    7
DmodifyTxAndTy =
    0
    0
    0
    0
tx,ty,DmodTxAndTy
tx =

```

```

    3
    ty =
    8
    DmodifyTxAndTy =
    0
    0
    0
    0
    tx,ty,DmodTxAndTy
    tx =
    3
    ty =
    9
    DmodifyTxAndTy =
    0
    0
    0
    0
    tx,ty,DmodTxAndTy
    tx =
    4
    ty =
    5
    DmodifyTxAndTy =
    0
    0
    0
    0
    tx,ty,DmodTxAndTy
    tx =
    4
    ty =
    6
    DmodifyTxAndTy =
    0
    0
    0
    0
    tx,ty,DmodTxAndTy
    tx =
    4
    ty =
    7
    DmodifyTxAndTy =
    0
    0
    0
    0
    tx,ty,DmodTxAndTy
    tx =
    4
    ty =
    8
    DmodifyTxAndTy =
    0
    0
    0
    0
    tx,ty,DmodTxAndTy
    tx =
    4
    ty =
    9
    DmodifyTxAndTy =
    0
    0
    0
    0
    tx,ty,DmodTxAndTy
    tx =
    5

```

```

ty =
    6
DmodifyTxAndTy =
    0
    0
    1
    0
id,is,tx,ty,rPx,rPy
id =
    3
is =
    1
tx =
    5
ty =
    6
rPx =
    0
rPy =
    0
id,is,tx,ty,rPx,rPy
id =
    3
is =
    2
tx =
    5
ty =
    6
rPx =
    0
rPy =
    0
id,is,tx,ty,rPx,rPy
id =
    3
is =
    3
tx =
    5
ty =
    6
rPx =
    0
rPy =
    0
id,is,tx,ty,rPx,rPy
id =
    3
is =
    4
tx =
    5
ty =
    6
rPx =
    0
rPy =
    0
tx,ty,DmodTxAndTy
tx =
    5
ty =
    7
DmodifyTxAndTy =
    0
    0
    0
    0
tx,ty,DmodTxAndTy
tx =

```

```

    5
    ty =
    8
    DmodifyTxAndTy =
    0
    0
    0
    0
    tx,ty,DmodTxAndTy
    tx =
    5
    ty =
    9
    DmodifyTxAndTy =
    0
    0
    0
    0
    tx,ty,DmodTxAndTy
    tx =
    6
    ty =
    7
    DmodifyTxAndTy =
    0
    0
    0
    0
    tx,ty,DmodTxAndTy
    tx =
    6
    ty =
    8
    DmodifyTxAndTy =
    0
    0
    0
    0
    tx,ty,DmodTxAndTy
    tx =
    6
    ty =
    9
    DmodifyTxAndTy =
    0
    0
    0
    0
    tx,ty,DmodTxAndTy
    tx =
    7
    ty =
    8
    DmodifyTxAndTy =
    0
    0
    0
    0
    tx,ty,DmodTxAndTy
    tx =
    7
    ty =
    9
    DmodifyTxAndTy =
    0
    0
    0
    0
    tx,ty,DmodTxAndTy
    tx =
    8

```

```

ty =
  9
DmodifyTxAndTy =
  0
  0
  0
  1
id,is,tx,ty,rPx,rPy
id =
  4
is =
  1
tx =
  8
ty =
  9
rPx =
  0
rPy =
  0
id,is,tx,ty,rPx,rPy
id =
  4
is =
  2
tx =
  8
ty =
  9
rPx =
  0
rPy =
  0
id,is,tx,ty,rPx,rPy
id =
  4
is =
  3
tx =
  8
ty =
  9
rPx =
  0
rPy =
  0
id,is,tx,ty,rPx,rPy
id =
  4
is =
  4
tx =
  8
ty =
  9
rPx =
  0
rPy =
  4
TsequenceRule2T =
  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  1
  0  0  0  0  0  0  0  0  0
isHomog =
  1

```

```

fileTR2T =
YSTsequenceRule2T.tgf
File YSTsequenceRule2T.tgf opened
nRows =
    9
nCols =
    9
File YSTsequenceRule2T.tgf closed
TsequenceRule1Plus2T =
    0    1    1    1    0    0    1    0    0
    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0
    0    0    0    0    1    1    1    0    0
    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    1    1
    0    0    0    0    0    0    0    0    1
    0    0    0    0    0    0    0    0    0
isHomog =
    1
fileTR12T =
YSTsequenceRule1Plus2T.tgf
File YSTsequenceRule1Plus2T.tgf opened
nRows =
    9
nCols =
    9
File YSTsequenceRule1Plus2T.tgf closed

```

Task-Task relation by Rule 3

Rule 3 for task sequence

```

TsequenceRule3T=zeros(nTask,nTask);
% Determine data modified by task x task y combinations
for tx=1:nTask-1
    for ty=tx+1:nTask
        % extract modification datasets for tasks and combine with 'and'
        DmodifyTx=DmodifiedByT(:,tx);
        DcreateTy=DcreatedByT(:,ty);
        DreadTy=DreadByT(:,ty);
        DmodifyTy=DmodifiedByT(:,ty);
        DmodifyTxAndReadTy=intersectionR(DmodifyTx,DreadTy);
        X=DmodifyTxAndReadTy;
        DcreateTyOrModifyTy=unionR(DcreateTy,DmodifyTy);
        Y=DcreateTyOrModifyTy;
        display 'tx,ty,X,Y',tx,ty,X,Y
        % check status ranks
        for id=1:nDataM
            if(X(id) == 1)
                rminX=nStatus
            for is=1:nStatus
                if(TwritesDS{id}(tx,is))==1
                    if(is<rminX) rminX=is ; end
                end
            end
        end
        if(Y(id) == 1)
            rmaxY=0
        for is=1:nStatus
            if(TwritesDS{id}(ty,is))==1
                if(is>rmaxY) rmaxY=is ; end
            end
        end
        if(rminX<=rmaxY) TsequenceRule2T(tx,ty)=1; end
    end
end
end
end

```

```

        end % loop over ty
    end % loop over tx
TsequenceRule3T

```

```
tx,ty,X,Y
```

```
tx =
```

```
1
```

```
ty =
```

```
2
```

```
X =
```

```
0
```

```
0
```

```
0
```

```
0
```

```
Y =
```

```
0
```

```
1
```

```
0
```

```
0
```

```
tx,ty,X,Y
```

```
tx =
```

```
1
```

```
ty =
```

```
3
```

```
X =
```

```
0
```

```
0
```

```
0
```

```
0
```

```
Y =
```

```
0
```

```
1
```

```
0
```

```
0
```

```
tx,ty,X,Y
```

```
tx =
```

```
1
```

```
ty =
```

```
4
```

```
X =
```

```
0
```

```
0
```

```
0
```

```
0
```

```
Y =
```

```
0
```

```
0
```

```
1
```

```
0
```

```
tx,ty,X,Y
```

```
tx =
```

```
1
```

```
ty =
```

```
5
```

```
X =
```

```
0
```

```
0
```

```
0
```

```
0
```

```
Y =
```

```
0
```

```
0
```

```
1
```

```
0
```

```
tx,ty,X,Y
```

```
tx =
```

```
1
```

```
ty =
```



```

        6
X =
    0
    0
    0
    0
Y =
    0
    0
    1
    0
tx,ty,X,Y
tx =
    1
ty =
    7
X =
    0
    0
    0
    0
Y =
    0
    0
    0
    1
tx,ty,X,Y
tx =
    1
ty =
    8
X =
    0
    0
    0
    0
Y =
    0
    0
    0
    1
tx,ty,X,Y
tx =
    1
ty =
    9
X =
    0
    0
    0
    0
Y =
    0
    0
    0
    1
tx,ty,X,Y
tx =
    2
ty =
    3
X =
    0
    1
    0
    0
Y =
    0
    1
    0
    0

```

```

rminX =
    4
rmaxY =
    0
tx,ty,X,Y
tx =
    2
ty =
    4
X =
    0
    1
    0
    0
Y =
    0
    0
    1
    0
rminX =
    4
tx,ty,X,Y
tx =
    2
ty =
    5
X =
    0
    0
    0
    0
Y =
    0
    0
    1
    0
tx,ty,X,Y
tx =
    2
ty =
    6
X =
    0
    0
    0
    0
Y =
    0
    0
    1
    0
tx,ty,X,Y
tx =
    2
ty =
    7
X =
    0
    1
    0
    0
Y =
    0
    0
    0
    1
rminX =
    4
tx,ty,X,Y
tx =
    2

```

```

ty =
  8
X =
  0
  0
  0
  0
Y =
  0
  0
  0
  1
tx,ty,X,Y
tx =
  2
ty =
  9
X =
  0
  0
  0
  0
Y =
  0
  0
  0
  1
tx,ty,X,Y
tx =
  3
ty =
  4
X =
  0
  1
  0
  0
Y =
  0
  0
  1
  0
rminX =
  4
tx,ty,X,Y
tx =
  3
ty =
  5
X =
  0
  0
  0
  0
Y =
  0
  0
  1
  0
tx,ty,X,Y
tx =
  3
ty =
  6
X =
  0
  0
  0
  0
Y =
  0

```

```

0
1
0
tx,ty,X,Y
tx =
3
ty =
7
X =
0
1
0
0
Y =
0
0
0
1
rminX =
4
tx,ty,X,Y
tx =
3
ty =
8
X =
0
0
0
0
Y =
0
0
0
1
tx,ty,X,Y
tx =
3
ty =
9
X =
0
0
0
0
Y =
0
0
0
1
tx,ty,X,Y
tx =
4
ty =
5
X =
0
0
0
0
Y =
0
0
1
0
tx,ty,X,Y
tx =
4
ty =
6
X =

```

```

0
0
0
0
Y =
0
0
1
0
tx,ty,X,Y
tx =
4
ty =
7
X =
0
0
0
0
Y =
0
0
0
1
tx,ty,X,Y
tx =
4
ty =
8
X =
0
0
0
0
Y =
0
0
0
1
tx,ty,X,Y
tx =
4
ty =
9
X =
0
0
0
0
Y =
0
0
0
1
tx,ty,X,Y
tx =
5
ty =
6
X =
0
0
1
0
Y =
0
0
1
0
rminX =
4

```

```

rmaxY =
  0
tx,ty,X,Y
tx =
  5
ty =
  7
X =
  0
  0
  1
  0
Y =
  0
  0
  0
  1
rminX =
  4
tx,ty,X,Y
tx =
  5
ty =
  8
X =
  0
  0
  0
  0
Y =
  0
  0
  0
  1
tx,ty,X,Y
tx =
  5
ty =
  9
X =
  0
  0
  0
  0
Y =
  0
  0
  0
  1
tx,ty,X,Y
tx =
  6
ty =
  7
X =
  0
  0
  1
  0
Y =
  0
  0
  0
  1
rminX =
  4
tx,ty,X,Y
tx =
  6
ty =
  8

```

```

X =
    0
    0
    0
    0
Y =
    0
    0
    0
    1
tx,ty,X,Y
tx =
    6
ty =
    9
X =
    0
    0
    0
    0
Y =
    0
    0
    0
    1
tx,ty,X,Y
tx =
    7
ty =
    8
X =
    0
    0
    0
    0
Y =
    0
    0
    0
    1
tx,ty,X,Y
tx =
    7
ty =
    9
X =
    0
    0
    0
    0
Y =
    0
    0
    0
    1
tx,ty,X,Y
tx =
    8
ty =
    9
X =
    0
    0
    0
    1
Y =
    0
    0
    0
    1
rminX =

```

```

4
rmaxY =
0
TsequenceRule3T =
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0

```

Combine Rules 1, 2 and 3 and determine task logical sequence

```

TsequenceT=unionR(unionR(TsequenceRule1T,TsequenceRule2T),TsequenceRule3T)
vertexLabels=horzcat(taskLabels',{'Tasks sequence with Tasks - Rule 1&2&3'});
isHomog=1
fileTT=horzcat(tgfFilePre,'TsequenceT','.tgf')
tgfWrite(fileTT,TsequenceT,isHomog,vertexLabels,{})
[Taskschedule]=TopolSortBFS(TsequenceT)
nSteps=size(Taskschedule,2)
stepLabel='Step';
for istep=1:nSteps
stepLabels{istep}=strcat(stepLabel,num2str(istep));
end
vertexLabels=horzcat(taskLabels,stepLabels',{'Tasks - Logical Steps'});
isHomog=0
fileTSchedule=horzcat(tgfFilePre,'Taskschedule','.tgf')
tgfWrite(fileTSchedule,Taskschedule,isHomog,vertexLabels,{})

```

```

TsequenceT =
0 1 1 1 0 0 1 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0

isHomog =
1
fileTT =
YSTsequenceT.tgf
File YSTsequenceT.tgf opened
nRows =
9
nCols =
9
File YSTsequenceT.tgf closed
T =
0 1 1 1 0 0 1 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0

***** while loop begin *****
After successor determination
After removals from A and B *****

```



```

After removals from A and C *****
TaskSchedule =
  1
  0
  0
  0
  0
  0
  0
  0
  0
  0
level =
  2
***** while loop begin *****
After successor determination
After removals from A and B ++++++
After removals from A and C *****
TaskSchedule =
  1    0
  0    1
  0    1
  0    1
  0    0
  0    0
  0    0
  0    0
  0    0
  0    0
level =
  3
***** while loop begin *****
After successor determination
After removals from A and B ++++++
After removals from A and C *****
TaskSchedule =
  1    0    0
  0    1    0
  0    1    0
  0    1    0
  0    0    1
  0    0    1
  0    0    1
  0    0    0
  0    0    0
level =
  4
***** while loop begin *****
After successor determination
After removals from A and B ++++++
After removals from A and C *****
TaskSchedule =
  1    0    0    0
  0    1    0    0
  0    1    0    0
  0    1    0    0
  0    0    1    0
  0    0    1    0
  0    0    1    0
  0    0    0    1
  0    0    0    0
level =
  5
***** while loop begin *****
After successor determination
After removals from A and B ++++++
After removals from A and C *****
TaskSchedule =
  1    0    0    0    0
  0    1    0    0    0
  0    1    0    0    0
  0    1    0    0    0
  0    0    1    0    0

```

```
    0    0    1    0    0
    0    0    1    0    0
    0    0    0    1    0
    0    0    0    0    1
level =
  6
CHasElements =
  0
Taskschedule =
  1    0    0    0    0
  0    1    0    0    0
  0    1    0    0    0
  0    1    0    0    0
  0    0    1    0    0
  0    0    1    0    0
  0    0    1    0    0
  0    0    0    1    0
  0    0    0    0    1
nSteps =
  5
isHomog =
  0
fileTSchedule =
YTaskschedule.tgf
File YTaskschedule.tgf opened
nRows =
  9
nCols =
  5
File YTaskschedule.tgf closed
```

Appendix J

Engineering Process Model Database

J.1 Eclipse Development software reference

The Eclipse SDK implementation of the Clay Database Modelling program was used to develop the database schema for the Engineering Process Model.

Reference to the software used is included here.

Eclipse SDK

Version: 3.1.2

Build id: M20060118-1600

(c) Copyright Eclipse contributors and others 2000, 2005. All rights reserved.
Visit <http://www.eclipse.org/platform>

This product includes software developed by the
Apache Software Foundation <http://www.apache.org/> Clay Database Modelling UI

Version: 1.2.0

Refer to The Eclipse Foundation [119]

J.2 Eclipse Azzurri Clay Eclipse Plugin for Database Modelling

Figure J.1 contains a detailed overview of the database tables as designed.

(c) Copyright Azzurri Ltd. All rights reserved.
Visit <http://www.azzurri.jp/>
Clay Core features enabled.

Refer to Azzurri Limited [10]

J.3 Azzurri Clay XML DTD Specification file

The well formedness can be checked by a program such as XMLSpy (Altova, Inc. [4] and the file validated using the DTD description below.

The structure of the .clay xml file is described in the DTD specification given below:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 3 U (http://www.xmlspy.com)
      by Breda Strasheim (Department of Civil Engineering) -->
<!--DTD generated by XMLSPY v5 rel. 3 U (http://www.xmlspy.com)-->
<!ELEMENT clay-model (database-model)>
```

```

<!ATTLIST clay-model
    clay-version CDATA #REQUIRED
>
<!ELEMENT column (column-description, data-type)>
<!ATTLIST column
    alias CDATA #REQUIRED
    auto-increment CDATA #REQUIRED
    column-size CDATA #REQUIRED
    decimal-digits CDATA #REQUIRED
    default-value CDATA #REQUIRED
    mandatory (false | true) #REQUIRED
    name CDATA #REQUIRED
    remarks CDATA #REQUIRED
>
<!ELEMENT column-description EMPTY>
<!ELEMENT column-list (column+)>
<!ELEMENT data-type (variant+)>
<!ATTLIST data-type
    jdbc-type (1 | 12 | 2) #REQUIRED
    literal-prefix CDATA #IMPLIED
    literal-suffix CDATA #IMPLIED
    name CDATA #REQUIRED
    selected-variant-pattern CDATA #REQUIRED
>
<!ELEMENT database-model (database-model-description, schema-list)>
<!ATTLIST database-model
    alias CDATA #REQUIRED
    author CDATA #REQUIRED
    begin-script CDATA #REQUIRED
    end-script CDATA #REQUIRED
    name CDATA #REQUIRED
    remarks CDATA #REQUIRED
    sql-dialect-id CDATA #REQUIRED
    version CDATA #REQUIRED
>
<!ELEMENT database-model-description EMPTY>
<!ELEMENT domain-list EMPTY>
<!ELEMENT fk-fig-bendpoint-list EMPTY>
<!ELEMENT foreign-key (foreign-key-description,
    foreign-key-figure, foreign-key-column)>
<!ATTLIST foreign-key
    alias CDATA #REQUIRED
    name CDATA #REQUIRED
    on-delete CDATA #REQUIRED
    on-update CDATA #REQUIRED
    referenced-key CDATA #REQUIRED
    referenced-table CDATA #REQUIRED
    referenced-table-schema CDATA #REQUIRED
    remarks CDATA #REQUIRED
    source-entity-role CDATA #REQUIRED
    source-multiplicity CDATA #REQUIRED
    source-relationship-type CDATA #REQUIRED
    target-entity-role CDATA #REQUIRED
    target-multiplicity CDATA #REQUIRED
    target-relationship-type CDATA #REQUIRED
>
<!ELEMENT foreign-key-column EMPTY>
<!ATTLIST foreign-key-column
    column-name CDATA #REQUIRED
    referenced-key-column-name CDATA #REQUIRED
>
<!ELEMENT foreign-key-description EMPTY>
<!ELEMENT foreign-key-figure (fk-fig-bendpoint-list)>
<!ELEMENT foreign-key-list (foreign-key*)>
<!ELEMENT index-list EMPTY>
<!ELEMENT primary-key (primary-key-description, primary-key-column)>
<!ATTLIST primary-key
    alias CDATA #REQUIRED
    name CDATA #REQUIRED
    remarks CDATA #REQUIRED
>
<!ELEMENT primary-key-column EMPTY>
<!ATTLIST primary-key-column
    name CDATA #REQUIRED
>
<!ELEMENT primary-key-description EMPTY>
<!ELEMENT schema (schema-description, domain-list, table-list)>
<!ATTLIST schema
    alias CDATA #REQUIRED
    name CDATA #REQUIRED
    remarks CDATA #REQUIRED
>
<!ELEMENT schema-description EMPTY>
<!ELEMENT schema-list (schema)>
<!ELEMENT table (table-description, table-figure-bounds,

```

```

        column-list, primary-key, unique-key-list,
                                foreign-key-list, index-list)>
<!--ATTLIST table
    alias CDATA #REQUIRED
    name CDATA #REQUIRED
    remarks CDATA #REQUIRED
-->
<!--ELEMENT table-description (#PCDATA)>
<!--ELEMENT table-figure-bounds EMPTY>
<!--ATTLIST table-figure-bounds
    height CDATA #REQUIRED
    width CDATA #REQUIRED
    x CDATA #REQUIRED
    y CDATA #REQUIRED
-->
<!--ELEMENT table-list (table+)>
<!--ELEMENT unique-key-list EMPTY>
<!--ELEMENT variant EMPTY>
<!--ATTLIST variant
    precision-max CDATA #IMPLIED
    precision-min CDATA #IMPLIED
    precision-variable CDATA #IMPLIED
    type-name-pattern CDATA #REQUIRED
    scale-max CDATA #IMPLIED
    scale-min CDATA #IMPLIED
    scale-variable CDATA #IMPLIED
-->

```

J.4 Database setup SQL statements

SQL statements generated ANSI 92 Standard used.

ON DELETE CASCADE added in second display below here - canClay handle this ??

```

-- Engineering Process Model Schema
-- EngProcessANSI-92.sql
DROP TABLE "tbEdgeDatasetTool";
DROP TABLE "tbEdgeTaskTool";
DROP TABLE "tbEdgeTaskDatasetModify";
DROP TABLE "tbEdgeTaskDatasetRead";
DROP TABLE "tbEdgeTaskTask";
DROP TABLE "tbEdgeTaskPerson";
DROP TABLE "tbEdgeTaskDatasetCreate";
DROP TABLE "tbTool";
DROP TABLE "tbTask";
DROP TABLE "tbDataset";
DROP TABLE "tbStatusElement";
DROP TABLE "tbAttribute";
DROP TABLE "tbPerson";

CREATE TABLE "tbPerson" (
    "PIDPerson" CHAR(10) NOT NULL
    , "PersonName" VARCHAR(100)
    , "PIDAttribute" CHAR(10)
    , "PIDStatusElement" CHAR(10)
    , PRIMARY KEY ("PIDPerson")
);

CREATE TABLE "tbAttribute" (
    "PIDAttribute" CHAR(10) NOT NULL
    , "AttributeName" VARCHAR(100)
    , "AttributeDiscipline" VARCHAR(100)
    , "AttributeFunction" VARCHAR(100)
    , "AttributeType" VARCHAR(100)
    , PRIMARY KEY ("PIDAttribute")
);

CREATE TABLE "tbStatusElement" (
    "PIDStatusElement" CHAR(10) NOT NULL
    , "StatusElementName" VARCHAR(100)
    , "Completion" NUMERIC(5,2)
    , PRIMARY KEY ("PIDStatusElement")
);

CREATE TABLE "tbDataset" (
    "PIDDataset" CHAR(10) NOT NULL
    , "DatasetName" VARCHAR(100)
    , "PIDAttribute" CHAR(10)
    , "PIDStatusElement" CHAR(10)
    , "Weight" NUMERIC(10,2)
    , PRIMARY KEY ("PIDDataset")

```

```

);

CREATE TABLE "tbTask" (
    "PIDTask" CHAR(10) NOT NULL
    , "TaskName" VARCHAR(100)
    , "LogicalStep" CHAR(10)
    , "PIDAttribute" CHAR(10)
    , "PIDStatusElement" CHAR(10)
    , PRIMARY KEY ("PIDTask")
);

CREATE TABLE "tbTool" (
    "PIDTool" CHAR(10) NOT NULL
    , "ToolName" VARCHAR(100)
    , "PIDAttribute" CHAR(10)
    , "PIDStatusElement" CHAR(10)
    , PRIMARY KEY ("PIDTool")
);

CREATE TABLE "tbEdgeTaskDatasetCreate" (
    "PIEdgeTaskDatasetCreate" CHAR(10) NOT NULL
    , "EdgeTaskDatasetCreateName" VARCHAR(100)
    , "PIDTask" CHAR(10)
    , "PIDDataset" CHAR(10)
    , "PIDAttribute" CHAR(10)
    , "PIDStatusElement" CHAR(10)
    , PRIMARY KEY ("PIEdgeTaskDatasetCreate")
);

CREATE TABLE "tbEdgeTaskPerson" (
    "PIEdgeTaskPerson" CHAR(10) NOT NULL
    , "EdgeTaskPersonName" VARCHAR(100)
    , "PIDTask" CHAR(10)
    , "PIDPerson" CHAR(10)
    , "PIDAttribute" CHAR(10)
    , "PIDStatusElement" CHAR(10)
    , PRIMARY KEY ("PIEdgeTaskPerson")
);

CREATE TABLE "tbEdgeTaskTask" (
    "PIEdgeTaskTask" CHAR(10) NOT NULL
    , "EdgeTaskTaskName" VARCHAR(100)
    , "PIDTaskIn" CHAR(10)
    , "PIDTaskOut" CHAR(10)
    , "PIDAttribute" CHAR(10)
    , "PIDStatusElement" CHAR(10)
    , "PIDTask" CHAR(10) NOT NULL
    , PRIMARY KEY ("PIEdgeTaskTask")
);

CREATE TABLE "tbEdgeTaskDatasetRead" (
    "PIEdgeTaskDatasetRead" CHAR(10) NOT NULL
    , "EdgeTaskDatasetReadName" VARCHAR(100)
    , "PIDTask" CHAR(10)
    , "PIDDataset" CHAR(10)
    , "PIDAttribute" CHAR(10)
    , "PIDStatusElement" CHAR(10)
    , PRIMARY KEY ("PIEdgeTaskDatasetRead")
);

CREATE TABLE "tbEdgeTaskDatasetModify" (
    "PIEdgeTaskDatasetModify" CHAR(10) NOT NULL
    , "EdgeTaskDatasetModifyName" VARCHAR(100)
    , "PIDTask" CHAR(10)
    , "PIDDataset" CHAR(10)
    , "PIDAttribute" CHAR(10)
    , "PIDStatusElement" CHAR(10)
    , PRIMARY KEY ("PIEdgeTaskDatasetModify")
);

CREATE TABLE "tbEdgeTaskTool" (
    "PIEdgeTaskTool" CHAR(10) NOT NULL
    , "EdgeTaskToolName" VARCHAR(100)
    , "PIDTask" CHAR(10)
    , "PIDTool" CHAR(10)
    , "PIDAttribute" CHAR(10)
    , "PIDStatusElement" CHAR(10)
    , PRIMARY KEY ("PIEdgeTaskTool")
);

CREATE TABLE "tbEdgeDatasetTool" (
    "PIEdgeDatasetTool" CHAR(10) NOT NULL
    , "EdgeDatasetToolName" VARCHAR(100)
    , "PIDDataset" CHAR(10)
    , "PIDTool" CHAR(10)

```

```

        , "PIDAttribute" CHAR(10)
        , "PIDStatusElement" CHAR(10)
        , PRIMARY KEY ("PIDEdgeDatasetTool")
    );

ALTER TABLE "tbTask"
    ADD CONSTRAINT "fkTask_1"
        FOREIGN KEY ("PIDAttribute")
        REFERENCES "tbAttribute" ("PIDAttribute");

ALTER TABLE "tbTask"
    ADD CONSTRAINT "fkTask_2"
        FOREIGN KEY ("PIDStatusElement")
        REFERENCES "tbStatusElement" ("PIDStatusElement");

ALTER TABLE "tbTool"
    ADD CONSTRAINT "FK_tbTool_1"
        FOREIGN KEY ("PIDStatusElement")
        REFERENCES "tbStatusElement" ("PIDStatusElement");

ALTER TABLE "tbEdgeTaskDatasetCreate"
    ADD CONSTRAINT "fkEdgeTaskDataset2"
        FOREIGN KEY ("PIDDataset")
        REFERENCES "tbDataset" ("PIDDataset");

ALTER TABLE "tbEdgeTaskDatasetCreate"
    ADD CONSTRAINT "fkEdgeTaskDataset4"
        FOREIGN KEY ("PIDAttribute")
        REFERENCES "tbAttribute" ("PIDAttribute");

ALTER TABLE "tbEdgeTaskDatasetCreate"
    ADD CONSTRAINT "fkEdgeTaskDataset6"
        FOREIGN KEY ("PIDStatusElement")
        REFERENCES "tbStatusElement" ("PIDStatusElement");

ALTER TABLE "tbEdgeTaskDatasetCreate"
    ADD CONSTRAINT "fkEdgeTaskDataset1"
        FOREIGN KEY ("PIDTask")
        REFERENCES "tbTask" ("PIDTask");

ALTER TABLE "tbEdgeTaskPerson"
    ADD CONSTRAINT "fkTaskPerson2"
        FOREIGN KEY ("PIDTask")
        REFERENCES "tbTask" ("PIDTask");

ALTER TABLE "tbEdgeTaskPerson"
    ADD CONSTRAINT "fkTaskPerson5"
        FOREIGN KEY ("PIDAttribute")
        REFERENCES "tbAttribute" ("PIDAttribute");

ALTER TABLE "tbEdgeTaskPerson"
    ADD CONSTRAINT "fkTaskPerson6"
        FOREIGN KEY ("PIDStatusElement")
        REFERENCES "tbStatusElement" ("PIDStatusElement");

ALTER TABLE "tbEdgeTaskPerson"
    ADD CONSTRAINT "fkTaskPerson3"
        FOREIGN KEY ("PIDPerson")
        REFERENCES "tbPerson" ("PIDPerson");

ALTER TABLE "tbEdgeTaskTask"
    ADD CONSTRAINT "fkEdgeTaskTask1"
        FOREIGN KEY ("PIDTask")
        REFERENCES "tbTask" ("PIDTask");

ALTER TABLE "tbEdgeTaskTask"
    ADD CONSTRAINT "fkEdgeTaskTask2"
        FOREIGN KEY ("PIDStatusElement")
        REFERENCES "tbStatusElement" ("PIDStatusElement");

ALTER TABLE "tbEdgeTaskTask"
    ADD CONSTRAINT "fkEdgeTaskTask3"
        FOREIGN KEY ("PIDAttribute")
        REFERENCES "tbAttribute" ("PIDAttribute");

ALTER TABLE "tbEdgeTaskDatasetRead"
    ADD CONSTRAINT "fkEdgeTaskDatasetRead1"
        FOREIGN KEY ("PIDTask")
        REFERENCES "tbTask" ("PIDTask");

ALTER TABLE "tbEdgeTaskDatasetRead"
    ADD CONSTRAINT "fkEdgeTaskDatasetRead2"
        FOREIGN KEY ("PIDDataset")
        REFERENCES "tbDataset" ("PIDDataset");

```

```

ALTER TABLE "tbEdgeTaskDatasetRead"
  ADD CONSTRAINT "fkEdgeTaskDatasetRead3"
    FOREIGN KEY ("PIDAttribute")
      REFERENCES "tbAttribute" ("PIDAttribute");

ALTER TABLE "tbEdgeTaskDatasetModify"
  ADD CONSTRAINT "fkEdgeTaskDatasetWrite1"
    FOREIGN KEY ("PIDTask")
      REFERENCES "tbTask" ("PIDTask");

ALTER TABLE "tbEdgeTaskDatasetModify"
  ADD CONSTRAINT "fkEdgeTaskDatasetWrite2"
    FOREIGN KEY ("PIDDataset")
      REFERENCES "tbDataset" ("PIDDataset");

ALTER TABLE "tbEdgeTaskDatasetModify"
  ADD CONSTRAINT "fkEdgeTaskDatasetWrite3"
    FOREIGN KEY ("PIDAttribute")
      REFERENCES "tbAttribute" ("PIDAttribute");

ALTER TABLE "tbEdgeTaskTool"
  ADD CONSTRAINT "fkEdgeTaskTool_4"
    FOREIGN KEY ("PIDStatusElement")
      REFERENCES "tbStatusElement" ("PIDStatusElement");

ALTER TABLE "tbEdgeTaskTool"
  ADD CONSTRAINT "fkEdgeTaskTool_1"
    FOREIGN KEY ("PIDTask")
      REFERENCES "tbTask" ("PIDTask");

ALTER TABLE "tbEdgeTaskTool"
  ADD CONSTRAINT "fkEdgeTaskTool_2"
    FOREIGN KEY ("PIDTool")
      REFERENCES "tbTool" ("PIDTool");

ALTER TABLE "tbEdgeTaskTool"
  ADD CONSTRAINT "fkEdgeTaskTool_3"
    FOREIGN KEY ("PIDAttribute")
      REFERENCES "tbAttribute" ("PIDAttribute");

ALTER TABLE "tbEdgeDatasetTool"
  ADD CONSTRAINT "fkEdgeDatasetTool4"
    FOREIGN KEY ("PIDStatusElement")
      REFERENCES "tbStatusElement" ("PIDStatusElement");

ALTER TABLE "tbEdgeDatasetTool"
  ADD CONSTRAINT "fkEdgeDatasetTool3"
    FOREIGN KEY ("PIDAttribute")
      REFERENCES "tbAttribute" ("PIDAttribute");

ALTER TABLE "tbEdgeDatasetTool"
  ADD CONSTRAINT "fkEdgeDatasetTool1"
    FOREIGN KEY ("PIDTool")
      REFERENCES "tbTool" ("PIDTool");

ALTER TABLE "tbEdgeDatasetTool"
  ADD CONSTRAINT "fkEdgeDatasetTool2"
    FOREIGN KEY ("PIDDataset")
      REFERENCES "tbDataset" ("PIDDataset");

```

```

-- Engineering Process Model Schema
-- EngProcessANSI-92.sql

DROP TABLE "tbEdgeDatasetTool";
DROP TABLE "tbEdgeTaskTool";
DROP TABLE "tbEdgeTaskDatasetModify";
DROP TABLE "tbEdgeTaskDatasetRead";
DROP TABLE "tbEdgeTaskTask";
DROP TABLE "tbEdgeTaskPerson";
DROP TABLE "tbEdgeTaskDatasetCreate";
DROP TABLE "tbTool";
DROP TABLE "tbTask";
DROP TABLE "tbDataset";
DROP TABLE "tbStatusElement";
DROP TABLE "tbAttribute";
DROP TABLE "tbPerson";

CREATE TABLE "tbPerson" (
  "PIDPerson" CHAR(10) NOT NULL
    , "PersonName" VARCHAR(100)
    , "PIDAttribute" CHAR(10)
    , "PIDStatusElement" CHAR(10)

```



```

, PRIMARY KEY ("PIDPerson")
);

CREATE TABLE "tbAttribute" (
  "PIDAttribute" CHAR(10) NOT NULL
, "AttributeName" VARCHAR(100)
, "AttributeDiscipline" VARCHAR(100)
, "AttributeFunction" VARCHAR(100)
, "AttributeType" VARCHAR(100)
, PRIMARY KEY ("PIDAttribute")
);

CREATE TABLE "tbStatusElement" (
  "PIDStatusElement" CHAR(10) NOT NULL
, "StatusElementName" VARCHAR(100)
, "Completion" NUMERIC(5,2)
, PRIMARY KEY ("PIDStatusElement")
);

CREATE TABLE "tbDataset" (
  "PIDDataset" CHAR(10) NOT NULL
, "DatasetName" VARCHAR(100)
, "PIDAttribute" CHAR(10)
, "PIDStatusElement" CHAR(10)
, "Weight" NUMERIC(10,2)
, PRIMARY KEY ("PIDDataset")
);

CREATE TABLE "tbTask" (
  "PIDTask" CHAR(10) NOT NULL
, "TaskName" VARCHAR(100)
, "LogicalStep" CHAR(10)
, "PIDAttribute" CHAR(10)
, "PIDStatusElement" CHAR(10)
, PRIMARY KEY ("PIDTask")
);

CREATE TABLE "tbTool" (
  "PIDTool" CHAR(10) NOT NULL
, "ToolName" VARCHAR(100)
, "PIDAttribute" CHAR(10)
, "PIDStatusElement" CHAR(10)
, PRIMARY KEY ("PIDTool")
);

CREATE TABLE "tbEdgeTaskDatasetCreate" (
  "PIEdgeTaskDatasetCreate" CHAR(10) NOT NULL
, "EdgeTaskDatasetCreateName" VARCHAR(100)
, "PIDTask" CHAR(10)
, "PIDDataset" CHAR(10)
, "PIDAttribute" CHAR(10)
, "PIDStatusElement" CHAR(10)
, PRIMARY KEY ("PIEdgeTaskDatasetCreate")
);

CREATE TABLE "tbEdgeTaskPerson" (
  "PIEdgeTaskPerson" CHAR(10) NOT NULL
, "EdgeTaskPersonName" VARCHAR(100)
, "PIDTask" CHAR(10)
, "PIDPerson" CHAR(10)
, "PIDAttribute" CHAR(10)
, "PIDStatusElement" CHAR(10)
, PRIMARY KEY ("PIEdgeTaskPerson")
);

CREATE TABLE "tbEdgeTaskTask" (
  "PIEdgeTaskTask" CHAR(10) NOT NULL
, "EdgeTaskTaskName" VARCHAR(100)
, "PIDTaskIn" CHAR(10)
, "PIDTaskOut" CHAR(10)
, "PIDAttribute" CHAR(10)
, "PIDStatusElement" CHAR(10)
, "PIDTask" CHAR(10) NOT NULL
, PRIMARY KEY ("PIEdgeTaskTask")
);

CREATE TABLE "tbEdgeTaskDatasetRead" (
  "PIEdgeTaskDatasetRead" CHAR(10) NOT NULL
, "EdgeTaskDatasetReadName" VARCHAR(100)
, "PIDTask" CHAR(10)
, "PIDDataset" CHAR(10)
, "PIDAttribute" CHAR(10)
, "PIDStatusElement" CHAR(10)
, PRIMARY KEY ("PIEdgeTaskDatasetRead")
);

```

```

CREATE TABLE "tbEdgeTaskDatasetModify" (
    "PIEdgeTaskDatasetModify" CHAR(10) NOT NULL
    , "EdgeTaskDatasetModifyName" VARCHAR(100)
    , "PIDTask" CHAR(10)
    , "PIDDataset" CHAR(10)
    , "PIDAttribute" CHAR(10)
    , "PIDStatusElement" CHAR(10)
    , PRIMARY KEY ("PIEdgeTaskDatasetModify")
);

CREATE TABLE "tbEdgeTaskTool" (
    "PIEdgeTaskTool" CHAR(10) NOT NULL
    , "EdgeTaskToolName" VARCHAR(100)
    , "PIDTask" CHAR(10)
    , "PIDTool" CHAR(10)
    , "PIDAttribute" CHAR(10)
    , "PIDStatusElement" CHAR(10)
    , PRIMARY KEY ("PIEdgeTaskTool")
);

CREATE TABLE "tbEdgeDatasetTool" (
    "PIEdgeDatasetTool" CHAR(10) NOT NULL
    , "EdgeDatasetToolName" VARCHAR(100)
    , "PIDDataset" CHAR(10)
    , "PIDTool" CHAR(10)
    , "PIDAttribute" CHAR(10)
    , "PIDStatusElement" CHAR(10)
    , PRIMARY KEY ("PIEdgeDatasetTool")
);

ALTER TABLE "tbTask"
    ADD CONSTRAINT "fkTask_1"
    FOREIGN KEY ("PIDAttribute")
    REFERENCES "tbAttribute" ("PIDAttribute")
    ON DELETE CASCADE;

ALTER TABLE "tbTask"
    ADD CONSTRAINT "fkTask_2"
    FOREIGN KEY ("PIDStatusElement")
    REFERENCES "tbStatusElement" ("PIDStatusElement")
    ON DELETE CASCADE;

ALTER TABLE "tbTool"
    ADD CONSTRAINT "FK_tbTool_1"
    FOREIGN KEY ("PIDStatusElement")
    REFERENCES "tbStatusElement" ("PIDStatusElement")
    ON DELETE CASCADE;

ALTER TABLE "tbEdgeTaskDatasetCreate"
    ADD CONSTRAINT "fkEdgeTaskDataset2"
    FOREIGN KEY ("PIDDataset")
    REFERENCES "tbDataset" ("PIDDataset")
    ON DELETE CASCADE;

ALTER TABLE "tbEdgeTaskDatasetCreate"
    ADD CONSTRAINT "fkEdgeTaskDataset4"
    FOREIGN KEY ("PIDAttribute")
    REFERENCES "tbAttribute" ("PIDAttribute")
    ON DELETE CASCADE;

ALTER TABLE "tbEdgeTaskDatasetCreate"
    ADD CONSTRAINT "fkEdgeTaskDataset6"
    FOREIGN KEY ("PIDStatusElement")
    REFERENCES "tbStatusElement" ("PIDStatusElement")
    ON DELETE CASCADE;

ALTER TABLE "tbEdgeTaskDatasetCreate"
    ADD CONSTRAINT "fkEdgeTaskDataset1"
    FOREIGN KEY ("PIDTask")
    REFERENCES "tbTask" ("PIDTask")
    ON DELETE CASCADE;

ALTER TABLE "tbEdgeTaskPerson"
    ADD CONSTRAINT "fkTaskPerson2"
    FOREIGN KEY ("PIDTask")
    REFERENCES "tbTask" ("PIDTask")
    ON DELETE CASCADE;

ALTER TABLE "tbEdgeTaskPerson"
    ADD CONSTRAINT "fkTaskPerson5"
    FOREIGN KEY ("PIDAttribute")
    REFERENCES "tbAttribute" ("PIDAttribute")
    ON DELETE CASCADE;

```

```

ALTER TABLE "tbEdgeTaskPerson"
  ADD CONSTRAINT "fkTaskPerson6"
    FOREIGN KEY ("PIDStatusElement")
      REFERENCES "tbStatusElement" ("PIDStatusElement")
    ON DELETE CASCADE;

ALTER TABLE "tbEdgeTaskPerson"
  ADD CONSTRAINT "fkTaskPerson3"
    FOREIGN KEY ("PIDPerson")
      REFERENCES "tbPerson" ("PIDPerson")
    ON DELETE CASCADE;

ALTER TABLE "tbEdgeTaskTask"
  ADD CONSTRAINT "fkEdgeTaskTask1"
    FOREIGN KEY ("PIDTask")
      REFERENCES "tbTask" ("PIDTask")
    ON DELETE CASCADE;

ALTER TABLE "tbEdgeTaskTask"
  ADD CONSTRAINT "fkEdgeTaskTask2"
    FOREIGN KEY ("PIDStatusElement")
      REFERENCES "tbStatusElement" ("PIDStatusElement")
    ON DELETE CASCADE;

ALTER TABLE "tbEdgeTaskTask"
  ADD CONSTRAINT "fkEdgeTaskTask3"
    FOREIGN KEY ("PIDAttribute")
      REFERENCES "tbAttribute" ("PIDAttribute")
    ON DELETE CASCADE;

ALTER TABLE "tbEdgeTaskDatasetRead"
  ADD CONSTRAINT "fkEdgeTaskDatasetRead1"
    FOREIGN KEY ("PIDTask")
      REFERENCES "tbTask" ("PIDTask")
    ON DELETE CASCADE;

ALTER TABLE "tbEdgeTaskDatasetRead"
  ADD CONSTRAINT "fkEdgeTaskDatasetRead2"
    FOREIGN KEY ("PIDDataset")
      REFERENCES "tbDataset" ("PIDDataset")
    ON DELETE CASCADE;

ALTER TABLE "tbEdgeTaskDatasetRead"
  ADD CONSTRAINT "fkEdgeTaskDatasetRead3"
    FOREIGN KEY ("PIDAttribute")
      REFERENCES "tbAttribute" ("PIDAttribute")
    ON DELETE CASCADE;

ALTER TABLE "tbEdgeTaskDatasetModify"
  ADD CONSTRAINT "fkEdgeTaskDatasetWrite1"
    FOREIGN KEY ("PIDTask")
      REFERENCES "tbTask" ("PIDTask")
    ON DELETE CASCADE;

ALTER TABLE "tbEdgeTaskDatasetModify"
  ADD CONSTRAINT "fkEdgeTaskDatasetWrite2"
    FOREIGN KEY ("PIDDataset")
      REFERENCES "tbDataset" ("PIDDataset")
    ON DELETE CASCADE;

ALTER TABLE "tbEdgeTaskDatasetModify"
  ADD CONSTRAINT "fkEdgeTaskDatasetWrite3"
    FOREIGN KEY ("PIDAttribute")
      REFERENCES "tbAttribute" ("PIDAttribute")
    ON DELETE CASCADE;

ALTER TABLE "tbEdgeTaskTool"
  ADD CONSTRAINT "fkEdgeTaskTool_4"
    FOREIGN KEY ("PIDStatusElement")
      REFERENCES "tbStatusElement" ("PIDStatusElement")
    ON DELETE CASCADE;

ALTER TABLE "tbEdgeTaskTool"
  ADD CONSTRAINT "fkEdgeTaskTool_1"
    FOREIGN KEY ("PIDTask")
      REFERENCES "tbTask" ("PIDTask")
    ON DELETE CASCADE;

ALTER TABLE "tbEdgeTaskTool"
  ADD CONSTRAINT "fkEdgeTaskTool_2"
    FOREIGN KEY ("PIDTool")
      REFERENCES "tbTool" ("PIDTool")
    ON DELETE CASCADE;

ALTER TABLE "tbEdgeTaskTool"

```

```

ADD CONSTRAINT "fkEdgeTaskTool_3"
FOREIGN KEY ("PIDAttribute")
REFERENCES "tbAttribute" ("PIDAttribute")
ON DELETE CASCADE;

ALTER TABLE "tbEdgeDatasetTool"
ADD CONSTRAINT "fkEdgeDatasetTool4"
FOREIGN KEY ("PIDStatusElement")
REFERENCES "tbStatusElement" ("PIDStatusElement")
ON DELETE CASCADE;

ALTER TABLE "tbEdgeDatasetTool"
ADD CONSTRAINT "fkEdgeDatasetTool3"
FOREIGN KEY ("PIDAttribute")
REFERENCES "tbAttribute" ("PIDAttribute")
ON DELETE CASCADE;

ALTER TABLE "tbEdgeDatasetTool"
ADD CONSTRAINT "fkEdgeDatasetTool1"
FOREIGN KEY ("PIDTool")
REFERENCES "tbTool" ("PIDTool")
ON DELETE CASCADE;

ALTER TABLE "tbEdgeDatasetTool"
ADD CONSTRAINT "fkEdgeDatasetTool2"
FOREIGN KEY ("PIDDataset")
REFERENCES "tbDataset" ("PIDDataset")
ON DELETE CASCADE;

```

J.5 PostgreSQL Database Reference

PostgreSQL 7.4.2 Documentation
The PostgreSQL Global Development Group

Refer to The PostgreSQL Global Development Group [121]

J.6 Engineering Process Model - Sample PostgreSQL Database Data Listing

The sample PostgreSQL database listing for the Engineering Process Model is given below. (Some field lengths need to be increased) ???

```
select * from public."tbPerson"
```

```

PIDPerson,PersonName

P0001,p01_Client
P0002,p02_Architect
P0003,p03_Structural Engineer
P0004,p04_Electrical Engineer
P0005,p05_Draftsman
P0006,p06_Checking Engineer

```

```
select * from public."tbTask"
```

```

PIDTask,TaskName,LogicalStep

T0001,t01_Create architectural design,Step0001
T0002,t02_Review architectural design,Step0002
T0003,t03_Preliminary structural design,Step0002
T0004,t04_Create foundation drawings,Step0003
T0005,t05_Create concrete layout drawings,Step0003
T0006,t06_Create reinforcement drawings,Step0004
T0007,t07_Preliminary electrical design,Step0002
T0008,t08_Create electrical drawings,Step0003

```

```

T0009,t09_Finalize architectural drawings,Step0003
T0010,t10_Finalize foundation drawings,Step0004
T0011,t11_Finalize concrete layout drawings,Step0004
T0012,t12_Finalize reinforcement drawings,Step0005
T0013,t13_Finalize electrical drawings,Step0004
T0014,t14_Finalize electrical design,Step0003
T0015,t15_Finalize structural design,Step0003
T0016,t16_Check structural design,Step0004
T0017,t17_Check architectural drawings,Step0004
T0018,t18_Check foundation drawings,Step0005
T0019,t19_Check concrete layout drawings,Step0005
T0020,t20_Check reinforcement drawings,Step0006
T0021,t21_Check electrical drawings,Step0005
T0022,t22_Check electrical design,Step0004

```

```
select * from public."tbTool"
```

```

PIDTool,ToolName

G0001,g01_CAD software
G0002,g02_Word processing software
G0003,g03_Spreadsheet software
G0004,g04_Hand calculations

```

```
select * from public."tbDataset"
```

```

PIDDataset,DatasetName,Weight,MilestoneClass

D0001,d01_Client requirements,(null),(null)
D0002,d02_Architectural drawings,(null),(null)
D0003,d03_Foundation drawings,(null),(null)
D0004,d04_Concrete layout drawings,(null),(null)
D0005,d05_Reinforcement drawings,(null),(null)
D0006,d06_Structural design report,(null),(null)
D0007,d07_Electrical drawings,(null),(null)
D0008,d08_Electrical design report,(null),(null)

```

```
select * from public."tbAttribute"
```

```

PIDAttribute,AttributeName,AttributeDiscipline,AttributeFunction,AttributeType

A0001,a01,Architectu,report,task
A0002,a02,Engineerin,time,person

```

```
select * from public."tbStatusElement"
```

```

PIDStatusElement,StatusElementName

S0001,assumed      s01
S0002,preliminary s02
S0003,engineered   s03
S0004,checked      s04

```

```
select * from public."tbEdgeTaskPerson"
```

```

PIDEdgeTaskPerson,EdgeTaskPersonName,PIDTask,PIDPerson,PIDAttribute,PIDStatusElement

T0001P0002,t01_Create architectural design-p02_Arch,T0001,P0002,(null),(null)
T0002P0001,t02_Review architectural design-p01_Clie,T0002,P0001,(null),(null)

```

```

T0002P0002,t02_Review architectural design-p02_Arch,T0002,P0002,(null),(null)
T0003P0003,t03_Preliminary structural design-p03_St,T0003,P0003,(null),(null)
T0005P0005,t05_Create concrete layout drawings-p05_,T0005,P0005,(null),(null)
T0006P0005,t06_Create reinforcement drawings-p05_Dr,T0006,P0005,(null),(null)
T0007P0005,t07_Preliminary electrical design-p05_Dr,T0007,P0005,(null),(null)
T0008P0005,t08_Create electrical drawings-p05_Draft,T0008,P0005,(null),(null)
T0009P0002,t09_Finalize architectural drawings-p02_,T0009,P0002,(null),(null)
T0009P0005,t09_Finalize architectural drawings-p05_,T0009,P0005,(null),(null)
T0010P0003,t10_Finalize foundation drawings-p03_Str,T0010,P0003,(null),(null)
T0010P0005,t10_Finalize foundation drawings-p05_Dra,T0010,P0005,(null),(null)
T0011P0002,t11_Finalize concrete layout drawings-p0,T0011,P0002,(null),(null)
T0011P0003,t11_Finalize concrete layout drawings-p0,T0011,P0003,(null),(null)
T0011P0005,t11_Finalize concrete layout drawings-p0,T0011,P0005,(null),(null)
T0012P0003,t12_Finalize reinforcement drawings-p03_,T0012,P0003,(null),(null)
T0012P0005,t12_Finalize reinforcement drawings-p05_,T0012,P0005,(null),(null)
T0013P0004,t13_Finalize electrical drawings-p04_Ele,T0013,P0004,(null),(null)
T0013P0005,t13_Finalize electrical drawings-p05_Dra,T0013,P0005,(null),(null)
T0014P0004,t14_Finalize electrical design-p04_Elect,T0014,P0004,(null),(null)
T0015P0003,t15_Finalize structural design-p03_Struc,T0015,P0003,(null),(null)
T0016P0006,t16_Check structural design-p06_Checking,T0016,P0006,(null),(null)
T0017P0006,t17_Check architectural drawings-p06_Che,T0017,P0006,(null),(null)
T0018P0006,t18_Check foundation drawings-p06_Checki,T0018,P0006,(null),(null)
T0019P0006,t19_Check concrete layout drawings-p06_C,T0019,P0006,(null),(null)
T0020P0006,t20_Check reinforcement drawings-p06_Che,T0020,P0006,(null),(null)
T0021P0006,t21_Check electrical drawings-p06_Checki,T0021,P0006,(null),(null)
T0022P0006,t22_Check electrical design-p06_Checking,T0022,P0006,(null),(null)

```

```
select * from public."tbEdgeTaskDataset"
```

```
PIDEdgeTaskDataset,EdgeTaskDatasetName,PIDTask,PIDDataset,PIDAttribute,PIDStatusElement
```

```

T0001D0002,t01_Create architectural design-d02_Arch,T0001,D0002,(null),S0001
T0003D0006,t03_Preliminary structural design-d06_St,T0003,D0006,(null),S0001
T0004D0003,t04_Create foundation drawings-d03_Found,T0004,D0003,(null),S0001
T0005D0004,t05_Create concrete layout drawings-d04_,T0005,D0004,(null),S0001
T0006D0005,t06_Create reinforcement drawings-d05_Re,T0006,D0005,(null),S0001
T0007D0008,t07_Preliminary electrical design-d08_El,T0007,D0008,(null),(null)
T0008D0007,t08_Create electrical drawings-d07_Elect,T0008,D0007,(null),S0001

```

```
select * from public."tbEdgeTaskTask"
```

```
PIDEdgeTaskTask,EdgeTaskTaskName,PIDTaskIn,PIDTaskOut,PIDAttribute,PIDStatusElement,PIDTask
```

```

T0001T0002,t01_Create architectural design-t02_Revi,T0001,T0002,(null),(null),T0001
T0001T0003,t01_Create architectural design-t03_Prel,T0001,T0003,(null),(null),T0001
T0001T0004,t01_Create architectural design-t04_Crea,T0001,T0004,(null),(null),T0001
T0001T0005,t01_Create architectural design-t05_Crea,T0001,T0005,(null),(null),T0001
T0001T0007,t01_Create architectural design-t07_Prel,T0001,T0007,(null),(null),T0001
T0001T0008,t01_Create architectural design-t08_Crea,T0001,T0008,(null),(null),T0001
T0001T0009,t01_Create architectural design-t09_Fina,T0001,T0009,(null),(null),T0001
T0001T0017,t01_Create architectural design-t17_Chec,T0001,T0017,(null),(null),T0001
T0002T0009,t02_Review architectural design-t09_Fina,T0002,T0009,(null),(null),T0002
T0002T0017,t02_Review architectural design-t17_Chec,T0002,T0017,(null),(null),T0002
T0003T0004,t03_Preliminary structural design-t04_Cr,T0003,T0004,(null),(null),T0003
T0003T0005,t03_Preliminary structural design-t05_Cr,T0003,T0005,(null),(null),T0003
T0003T0006,t03_Preliminary structural design-t06_Cr,T0003,T0006,(null),(null),T0003
T0003T0015,t03_Preliminary structural design-t15_Fi,T0003,T0015,(null),(null),T0003
T0003T0016,t03_Preliminary structural design-t16_Ch,T0003,T0016,(null),(null),T0003
T0004T0010,t04_Create foundation drawings-t10_Final,T0004,T0010,(null),(null),T0004
T0004T0018,t04_Create foundation drawings-t18_Check,T0004,T0018,(null),(null),T0004
T0005T0006,t05_Create concrete layout drawings-t06_,T0005,T0006,(null),(null),T0005
T0005T0011,t05_Create concrete layout drawings-t11_,T0005,T0011,(null),(null),T0005
T0005T0012,t05_Create concrete layout drawings-t12_,T0005,T0012,(null),(null),T0005
T0005T0019,t05_Create concrete layout drawings-t19_,T0005,T0019,(null),(null),T0005
T0006T0012,t06_Create reinforcement drawings-t12_Fi,T0006,T0012,(null),(null),T0006
T0006T0020,t06_Create reinforcement drawings-t20_Ch,T0006,T0020,(null),(null),T0006
T0007T0008,t07_Preliminary electrical design-t08_Cr,T0007,T0008,(null),(null),T0007

```

```
T0007T0014,t07_Preliminary electrical design-t14_Fi,T0007,T0014,(null),(null),T0007
T0007T0022,t07_Preliminary electrical design-t22_Ch,T0007,T0022,(null),(null),T0007
T0008T0013,t08_Create electrical drawings-t13_Final,T0008,T0013,(null),(null),T0008
T0008T0021,t08_Create electrical drawings-t21_Check,T0008,T0021,(null),(null),T0008
T0009T0017,t09_Finalize architectural drawings-t17_,T0009,T0017,(null),(null),T0009
T0010T0018,t10_Finalize foundation drawings-t18_Che,T0010,T0018,(null),(null),T0010
T0011T0019,t11_Finalize concrete layout drawings-t1,T0011,T0019,(null),(null),T0011
T0012T0020,t12_Finalize reinforcement drawings-t20_,T0012,T0020,(null),(null),T0012
T0013T0021,t13_Finalize electrical drawings-t21_Che,T0013,T0021,(null),(null),T0013
T0014T0022,t14_Finalize electrical design-t22_Check,T0014,T0022,(null),(null),T0014
T0015T0016,t15_Finalize structural design-t16_Check,T0015,T0016,(null),(null),T0015
```

```
select * from public."tbEdgeDatasetTool"
```

```
PIDEdgeDatasetTool,EdgeDatasetToolName,PIDDataset,PIDTool,PIDAttribute,PIDStatusElement

D0001G0002,d01_Client requirements-g02_Word process,D0001,G0002,(null),(null)
D0001G0003,d01_Client requirements-g03_Spreadsheet,D0001,G0003,(null),(null)
D0001G0004,d01_Client requirements-g04_Hand calcula,D0001,G0004,(null),(null)
D0002G0001,d02_Architectural drawings-g01_CAD softw,D0002,G0001,(null),(null)
D0003G0001,d03_Foundation drawings-g01_CAD software,D0003,G0001,(null),(null)
D0004G0001,d04_Concrete layout drawings-g01_CAD sof,D0004,G0001,(null),(null)
D0005G0001,d05_Reinforcement drawings-g01_CAD softw,D0005,G0001,(null),(null)
D0006G0002,d06_Structural design report-g02_Word pr,D0006,G0002,(null),(null)
D0006G0003,d06_Structural design report-g03_Spreads,D0006,G0003,(null),(null)
D0006G0004,d06_Structural design report-g04_Hand ca,D0006,G0004,(null),(null)
D0007G0001,d07_Electrical drawings-g01_CAD software,D0007,G0001,(null),(null)
D0008G0002,d08_Electrical design report-g02_Word pr,D0008,G0002,(null),(null)
D0008G0003,d08_Electrical design report-g03_Spreads,D0008,G0003,(null),(null)
```

J.7 Microsoft Access Database Reference

The Microsoft Access Database package was used demonstrate the Engineering Process model functionality in the desktop computer application environment on the Microsoft Windows XP operating system. Refer to the software reference below.

Application name	Microsoft Access
Version	10.0
Build	6771
Product ID	54185-640-0778382-17145
Application path	C:\Program Files\Microsoft Office\Office10\
Language	English (United States)
ADO version	Not Available
VBA version	6.04

Access programming techniques were drawn from Litwin et al. [75], Litwin et al. [76] and Litwin et al. [77].

J.8 Access data import process

The data is input via Access Table links using a VBA program genertaing SQL queries for the DAO subsystem of Access

J.8.1 Database schema definition file

The schema.ini file defines the record contents of the .csv data files per database table.

```
[PEPEExtbAttribute.csv]
ColNameHeader=False
Format=CSVDelimited
MaxScanRows=0
CharacterSet=OEM
Col1="PIDAttribute" Text
Col2="AttributeName" Text
Col3="AttributeDiscipline" Text
Col4="AttributeFunction" Text
Col5="AttributeType" Text

[PEPEExtbStatusElement.csv]
ColNameHeader=False
Format=CSVDelimited
MaxScanRows=0
CharacterSet=OEM
Col1="PIDStatusElement" Text
Col2="StatusElementName" Text
Col3="Completion" Long

[PEPEExtbDataset.csv]
ColNameHeader=False
Format=CSVDelimited
MaxScanRows=0
CharacterSet=OEM
Col1="PIDDataset" Text
Col2="DatasetName" Text
Col3="PIDAttribute" Text
Col4="PIDStatusElement" Text
Col5="Weight" Long

[PEPEExtbPerson.csv]
ColNameHeader=False
Format=CSVDelimited
MaxScanRows=0
CharacterSet=OEM
Col1="PIDPerson" Text
Col2="Personname" Text
Col3="PIDAttribute" Text
Col4="PIDStatusElement" Text

[PEPEExtbTask.csv]
ColNameHeader=False
Format=CSVDelimited
MaxScanRows=0
CharacterSet=OEM
Col1="PIDTask" Text
Col2="Taskname" Text
Col3="LogicalStep" Text
Col4="PIDAttribute" Text
Col5="PIDStatusElement" Text

[PEPEExtbTool.csv]
ColNameHeader=False
Format=CSVDelimited
MaxScanRows=0
CharacterSet=OEM
Col1="PIDTool" Text
Col2="ToolName" Text
Col3="PIDAttribute" Text
Col4="PIDStatusElement" Text

[PEPEExtbEdgeTaskDatasetCreate.csv]
ColNameHeader=False
Format=CSVDelimited
MaxScanRows=0
CharacterSet=OEM
Col1="PIDEdgeTaskDatasetCreate" Text
Col2="EdgeTaskDatasetCreateName" Text
Col3="PIDTask" Text
```



```

Col4="PIDDataset" Text
Col5="PIDAttribute" Text
Col6="PIDStatusElement" Text

[PEPEExtbEdgeTaskPerson.csv]
ColNameHeader=False
Format=CSVDelimited
MaxScanRows=0
CharacterSet=OEM
Col1="PIDEdgeTaskPerson" Text
Col2="EdgeTaskPersonName" Text
Col3="PIDTask" Text
Col4="PIDPerson" Text
Col5="PIDAttribute" Text
Col6="PIDStatusElement" Text

[PEPEExtbEdgeTaskTask.csv]
ColNameHeader=False
Format=CSVDelimited
MaxScanRows=0
CharacterSet=OEM
Col1="PIDEdgeTaskTask" Text
Col2="EdgeTaskTaskName" Text
Col3="PIDTaskIn" Text
Col4="PIDTaskOut" Text
Col5="PIDAttribute" Text
Col6="PIDStatusElement" Text
Col7="PIDTask" Text

[PEPEExtbEdgeTaskDatasetRead.csv]
ColNameHeader=False
Format=CSVDelimited
MaxScanRows=0
CharacterSet=OEM
Col1="PIDEdgeTaskDatasetRead" Text
Col2="EdgeTaskDatasetReadName" Text
Col3="PIDTask" Text
Col4="PIDDataset" Text
Col5="PIDAttribute" Text
Col6="PIDStatusElement" Text

[PEPEExtbEdgeTaskDatasetModify.csv]
ColNameHeader=False
Format=CSVDelimited
MaxScanRows=0
CharacterSet=OEM
Col1="PIDEdgeTaskDatasetModify" Text
Col2="EdgeTaskDatasetModifyName" Text
Col3="PIDTask" Text
Col4="PIDDataset" Text
Col5="PIDAttribute" Text
Col6="PIDStatusElement" Text

[PEPEExtbEdgeTaskTool.csv]
ColNameHeader=False
Format=CSVDelimited
MaxScanRows=0
CharacterSet=OEM
Col1="PIDEdgeTaskTool" Text
Col2="EdgeTaskToolName" Text
Col3="PIDTask" Text
Col4="PIDTool" Text
Col5="PIDAttribute" Text
Col6="PIDStatusElement" Text

[PEPEExtbEdgeDatasetTool.csv]
ColNameHeader=False
Format=CSVDelimited
MaxScanRows=0
CharacterSet=OEM

```

```

Col1="PIDEdgeDatasetTool" Text
Col2="EdgeDatasetToolName" Text
Col3="PIDDataset" Text
Col4="PIDTool" Text
Col5="PIDAttribute" Text
Col6="PIDStatusElement" Text

```

J.8.2 Access VBA code for data import

The Access VBA code to import the data into the database is given below:

```

Private Sub ImportData()
    Dim objAccessApp As Access.Application
    Set objAccessApp = GetObject(, "Access.Application")
    Dim filePrefix As String
    Dim tableName As String
    Dim tablePrefix As String
    filePrefix = "PEPEEx"
    tablePrefix = "public_"
,
' Delete entries edges first then vertices
    tableName = "tbEdgeTaskPerson"
    Call csvTableDataDelete(objAccessApp, filePrefix, tablePrefix, _
        tableName)
    tableName = "tbEdgeTaskTask"
    Call csvTableDataDelete(objAccessApp, filePrefix, tablePrefix, _
        tableName)
    tableName = "tbEdgeTaskDatasetCreate"
    Call csvTableDataDelete(objAccessApp, filePrefix, tablePrefix, _
        tableName)
    tableName = "tbEdgeTaskDatasetRead"
    Call csvTableDataDelete(objAccessApp, filePrefix, tablePrefix, _
        tableName)
    tableName = "tbEdgeTaskDatasetModify"
    Call csvTableDataDelete(objAccessApp, filePrefix, tablePrefix, _
        tableName)
    tableName = "tbEdgeTaskTool"
    Call csvTableDataDelete(objAccessApp, filePrefix, tablePrefix, _
        tableName)
    tableName = "tbEdgeDatasetTool"
    Call csvTableDataDelete(objAccessApp, filePrefix, tablePrefix, _
        tableName)
    tableName = "tbPerson"
    Call csvTableDataDelete(objAccessApp, filePrefix, tablePrefix, _
        tableName)
    tableName = "tbTask"
    Call csvTableDataDelete(objAccessApp, filePrefix, tablePrefix, _
        tableName)
    tableName = "tbAttribute"
    Call csvTableDataDelete(objAccessApp, filePrefix, tablePrefix, _
        tableName)
    tableName = "tbStatusElement"
    Call csvTableDataDelete(objAccessApp, filePrefix, tablePrefix, _
        tableName)
    tableName = "tbDataset"
    Call csvTableDataDelete(objAccessApp, filePrefix, tablePrefix, _
        tableName)
    tableName = "tbTool"
    Call csvTableDataDelete(objAccessApp, filePrefix, tablePrefix, _
        tableName)
,
' Import information - vertices first
    tableName = "tbPerson"
    Call csvTableDataImport(objAccessApp, filePrefix, tablePrefix, _
        tableName)
    tableName = "tbTask"
    Call csvTableDataImport(objAccessApp, filePrefix, tablePrefix, _
        tableName)

```

```

        tableName = "tbAttribute"
        Call csvTableDataImport(objAccessApp, filePrefix, tablePrefix, _
            tableName)
        tableName = "tbStatusElement"
        Call csvTableDataImport(objAccessApp, filePrefix, tablePrefix, _
            tableName)
        tableName = "tbDataset"
        Call csvTableDataImport(objAccessApp, filePrefix, tablePrefix, _
            tableName)
        tableName = "tbTool"
        Call csvTableDataImport(objAccessApp, filePrefix, tablePrefix, _
            tableName)
        tableName = "tbEdgeTaskPerson"
        Call csvTableDataImport(objAccessApp, filePrefix, tablePrefix, _
            tableName)
        tableName = "tbEdgeTaskTask"
        Call csvTableDataImport(objAccessApp, filePrefix, tablePrefix, _
            tableName)
        tableName = "tbEdgeTaskDatasetCreate"
        Call csvTableDataImport(objAccessApp, filePrefix, tablePrefix, _
            tableName)
        tableName = "tbEdgeTaskDatasetRead"
        Call csvTableDataImport(objAccessApp, filePrefix, tablePrefix, _
            tableName)
        tableName = "tbEdgeTaskDatasetModify"
        Call csvTableDataImport(objAccessApp, filePrefix, tablePrefix, _
            tableName)
        tableName = "tbEdgeTaskTool"
        Call csvTableDataImport(objAccessApp, filePrefix, tablePrefix, _
            tableName)
        tableName = "tbEdgeDatasetTool"
        Call csvTableDataImport(objAccessApp, filePrefix, tablePrefix, _
            tableName)

        Set objAccessApp = Nothing
    End Sub

Private Sub csvTableDataImport(objAccessApp As Application, _
    filePrefix As String, _
    tablePrefix As String, _
    tableName As String)

    Dim db As DAO.Database
    Set db = CurrentDb()
    Debug.Print objAccessApp.CurrentProject.Path
    MsgBox "Importing " & filePrefix & tableName & ".csv " & _
        Chr(13) & " into Table " & _
        tablePrefix & tableName & " using Schema.ini"
    db.Execute "DELETE FROM " & tablePrefix & tableName & ";"
    db.Execute _
        "INSERT INTO " & tablePrefix & tableName & _
        " SELECT * " & _
        " FROM [Text;FMT=Delimited;HDR=No;" & _
        " DATABASE=" & objAccessApp.CurrentProject.Path & _
        ";].[" & filePrefix & tableName & "#csv];"
    db.TableDefs.Refresh
End Sub

Private Sub csvTableDataDelete(objAccessApp As Application, _
    filePrefix As String, _
    tablePrefix As String, _
    tableName As String)

    Dim db As DAO.Database
    Set db = CurrentDb()
    Debug.Print objAccessApp.CurrentProject.Path
    MsgBox "Deleting records for " & _
        " Table " & _
        tablePrefix & tableName, _
        Buttons = vbInformation
    db.Execute "DELETE FROM " & tablePrefix & tableName & ";"
    db.TableDefs.Refresh
End Sub

```

J.8.3 Database program to import data from database server

Database Processing using Microsoft Access 1997/2002/2003

DDL queries are a subset of SQL queries which include:

```
CREATE TABLE
ALTER TABLE
CREATE INDEX
CREATE INDEX
DROP TABLE
DROP INDEX
```

Access can accept DDL type queries in its query SQL view but can only execute one SQL statement at a time! This is a major drawback in importing DDL data from database design tools such as Azzurri Clay.

As an alternative VBA programming using the ADO (ActiveX Data Object) library and class/object model can be used.

Refer to Litwin, Getz & Gunderloy - Access Developers Handbook Chapter 5 Litwin et al. [76].

Microsoft Windows Data links.

Microsoft Windows has a data link definition facility which can be used to set up a specification for a data link (data source) for use in Windows applications.

A blank text file is created using e.g. Windows Explorer, it is renamed to have a .udl extension and when the file is then opened it has the UDL functionality shown below.

DataLinks used to be MSDASC.dll, but has changed to OLEDB32.DLL

The link can then be used in VBA programs to access data without specifying the link parameters in the program code.

Access DLL Library references.

```
[oledb]
; Everything after this line is an OLE DB initstring
Provider=MSDASQL.1;
Persist Security Info=False;
User ID=java;
Data Source=PostgreSQLjava;
Initial Catalog=public
```

Sample Microsoft Access VBA Code.

```
Option Compare Database
Option Explicit

' From Access 2002 Desktop Developer's Handbook
' by Litwin, Getz, and Gunderloy. (Sybex)
' Copyright 2001. All rights reserved.

Public Sub TestDataLink()
    ' To set things up, run the
    ' ShowDataLink procedure before
    ' running this one.
    Dim cnn As ADODB.Connection
    Set cnn = New ADODB.Connection
    cnn.Open "File Name=" & _
        CurrentProject.Path & "\PSQLEngProcess.udl"
    Debug.Print cnn.ConnectionString
    Set cnn = Nothing
End Sub
```

```

Option Compare Database
Option Explicit
Public Sub ShowDataLink2()
    Dim cnn As ADODB.Connection
    'DataLinks used to be MSDASC.dll, but has changed to OLEDB32.DLL
    Dim dlk As MSDASC.DataLinks
    Set cnn = New ADODB.Connection
    Set dlk = New MSDASC.DataLinks
    ' Set default properties for the connection
    cnn.Provider = "Microsoft.Jet.OLEDB.4.0"
    cnn.Properties("Data Source") = _
        CurrentProject.Path & "\ODBCpSQLEngProcess.mdb"
    ' Tell the Data Link Properties dialog which
    ' window will be its parent
    dlk.Hwnd = Application.hWndAccessApp
    ' Prompt the user for information
    If dlk.PromptEdit(cnn) Then
        cnn.Open
    End If
    Set cnn = Nothing
    Set dlk = Nothing
End Sub

```

Refer to Litwin, Getz & Gunderloy - Access Developers Handbook Chapter 6, Litwin et al. [76].

Note that the ADOX will not work for all remote database server.

The Catalog / Table / Record structure need not be supported by all other Databases e.g. PostgreSQL on Linux.

Comment on PostgreSQL Web Site: <http://archives.postgresql.org/pgsql-odbc/2002-09/msg00049.php>

Microsoft ActiveX Data Objects Extensions for Data Definition Language and Security (ADOX) is designed for use with the Microsoft Jet Database Engine. So, using ADOX with OLE DB providers other than the Microsoft Jet OLE DB Provider may cause unexpected behavior or incorrect results. The exact behavior is dependent on the nature of the database for which the provider is written. If a provider accesses a database system whose model is totally different from that of Jet, the behavior of ADOX could be unpredictable (for example, Jet does not support the concepts of CATALOG or SCHEMA)....

<http://support.microsoft.com/default.aspx?scid=kb;>

Errors like the one below occur.

-2147352566

Use linked tables in this case.

J.9 PostgreSQL - Importing data into database

Using PSQL COPY FROM to import data.

Refer to The PostgreSQL Global Development Group, PSQL Documentation in The PostgreSQL Global Development Group [121].

COPY FROM

Use COPY FROM Use COPY FROM STDIN to load all the rows in one command, instead of using a series of INSERT commands. This reduces parsing, planning, etc. overhead a great deal. If you do this then it is not necessary to turn off autocommit, since it is only one command anyway.

```

COPY
Name
COPY- copy data between a file and a table
Synopsis
COPY tablename [ ( column [, ...] ) ]
FROM { 'filename' | STDIN }
[ [ WITH ]
[ BINARY ]
[ OIDS ]
[ DELIMITER [ AS ] 'delimiter' ]
[ NULL [ AS ] 'null string' ] ]
COPY tablename [ ( column [, ...] ) ]

```

```

TO { 'filename' | STDOUT }
[ [ WITH ]
[ BINARY ]
[ OIDS ]
[ DELIMITER [ AS ] 'delimiter' ]
[ NULL [ AS ] 'null string' ] ]

```

```

DELETE FROM "tbDataset";
COPY "tbDataset" FROM 'PEPEExtbDataset.csv' DELIMITER AS ,      ;

```

Only available to super users! The format below is an alternative for normal users Note that no colon terminates the statement

Delete / load / display records sequence:

```

DELETE FROM "tbDataset";
\copy "tbDataset" FROM 'PEPEExtbDataset.csv' DELIMITER AS ,
SELECT * FROM "tbDataset";

```

Special entries for bank attributes and status values were added to cater for vertices / edges without attributes or status values.

```

-- LoadData.psql -- A mixture of SQL and PSQL statements --

DELETE FROM "tbAttribute"; \copy "tbAttribute" FROM 'PEPEExtbAttribute.csv'
DELIMITER AS , SELECT * FROM "tbAttribute";

DELETE FROM "tbStatusElement"; \copy "tbStatusElement" FROM
'PEPEExtbStatusElement.csv' DELIMITER AS , SELECT * FROM "tbStatusElement";

DELETE FROM "tbDataset"; \copy "tbDataset" FROM 'PEPEExtbDataset.csv'
DELIMITER AS , SELECT * FROM "tbDataset";

DELETE FROM "tbPerson"; \copy "tbPerson" FROM 'PEPEExtbPerson.csv' DELIMITER
AS , SELECT * FROM "tbPerson";

DELETE FROM "tbTask"; \copy "tbTask" FROM 'PEPEExtbTask.csv' DELIMITER AS ,
SELECT * FROM "tbTask";

DELETE FROM "tbTool"; \copy "tbTool" FROM 'PEPEExtbTool.csv' DELIMITER AS ,
SELECT * FROM "tbPerson";

DELETE FROM "tbEdgeDatasetTool"; \copy "tbEdgeDatasetTool" FROM
'PEPEExtbEdgeDatasetTool.csv' DELIMITER AS , SELECT * FROM
"tbEdgeDatasetTool";

DELETE FROM "tbEdgeTaskDatasetCreate"; \copy "tbEdgeTaskDatasetCreate" FROM
'PEPEExtbEdgeTaskDatasetCreate.csv' DELIMITER AS , SELECT * FROM
"tbEdgeTaskDatasetCreate";

DELETE FROM "tbEdgeTaskDatasetModify"; \copy "tbEdgeTaskDatasetModify" FROM
'PEPEExtbEdgeTaskDatasetModify.csv' DELIMITER AS , SELECT * FROM
"tbEdgeTaskDatasetModify";

DELETE FROM "tbEdgeTaskDatasetRead"; \copy "tbEdgeTaskDatasetRead" FROM
'PEPEExtbEdgeTaskDatasetRead.csv' DELIMITER AS , SELECT * FROM
"tbEdgeTaskDatasetRead";

DELETE FROM "tbEdgeTaskPerson"; \copy "tbEdgeTaskPerson" FROM
'PEPEExtbEdgeTaskPerson.csv' DELIMITER AS , SELECT * FROM "tbEdgeTaskPerson";

DELETE FROM "tbEdgeTaskTask"; \copy "tbEdgeTaskTask" FROM
'PEPEExtbEdgeTaskTask.csv' DELIMITER AS , SELECT * FROM "tbEdgeTaskTask";

DELETE FROM "tbEdgeTaskTool"; \copy "tbEdgeTaskTool" FROM
'PEPEExtbEdgeTaskTool.csv' DELIMITER AS , SELECT * FROM "tbEdgeTaskTool";

```

Output generated by running .psql file i.e.: \i LoadData.psql

DELETE 3

PIDAttribute	AttributeName	AttributeDiscipline	AttributeFunction	AttributeType
A0001	a01	Architecture	design	drawing
A0002	a02	Structural Engineering	drafting	report
A0003	noAttributeName	noAttributeDiscipline	noAttributeFunction	noAttributeType

(3 rows)

DELETE 3				
PIDAttribute	AttributeName	AttributeDiscipline	AttributeFunction	AttributeType
A0001	a01	Architecture	design	drawing
A0002	a02	Structural Engineering	drafting	report
A0003	noAttributeName	noAttributeDiscipline	noAttributeFunction	noAttributeType
(3 rows)				

hrule
hrulefill
rule

DELETE 3

PIDAttribute	AttributeName	AttributeDiscipline	AttributeFunction	AttributeType
A0001	a01	Architecture	design	drawing
A0002	a02	Structural Engineering	drafting	report
A0003	noAttributeName	noAttributeDiscipline	noAttributeFunction	noAttributeType

(3 rows)

DELETE 3

PIDAttribute	AttributeName	AttributeDiscipline	AttributeFunction	AttributeType
A0001	a01	Architecture	design	drawing
A0002	a02	Structural Engineering	drafting	report
A0003	noAttributeName	noAttributeDiscipline	noAttributeFunction	noAttributeType

(3 rows)

hrule

hrulefill

rule

DELETE 3

PIDAttribute	AttributeName	AttributeDiscipline	AttributeFunction	AttributeType
A0001	a01	Architecture	design	drawing
A0002	a02	Structural Engineering	drafting	report
A0003	noAttributeName	noAttributeDiscipline	noAttributeFunction	noAttributeType

(3 rows)

DELETE 13

PIDStatusElement	StatusElementName	Completion
S0001	se01_Architectural Drawings - Assumed	20.00
S0002	se02_Architectural Drawings - Preliminary	20.00
S0003	se03_Architectural Drawings - Engineered	50.00
S0004	se04_Architectural Drawings - Checked	10.00
S0005	se05_Engineering Drawings - Assumed	20.00
S0006	se06_Engineering Drawings - Preliminary	0.00
S0007	se07_Engineering Drawings - Engineered	75.00
S0008	se08_Engineering Drawings - Checked	5.00
S0009	se09_Engineering Designs - Assumed	20.00
S0010	se10_Engineering Designs - Preliminary	0.00
S0011	se11_Engineering Designs - Engineered	70.00
S0012	se12_Engineering Designs - Checked	10.00
S0013	noStatusElementName	0.00

(13 rows)

DELETE 8

PIDDataset	DatasetName	PIDAttribute	PIDStatusElement	Weight
D0001	d01_Client requirements	A0003	S0013	0.00
D0002	d02_Architectural drawings	A0003	S0013	25.00
D0003	d03_Foundation drawings	A0003	S0013	10.00
D0004	d04_Concrete layout drawings	A0003	S0013	50.00
D0005	d05_Reinforcement drawings	A0003	S0013	12.00
D0006	d06_Structural design report	A0003	S0013	25.00
D0007	d07_Electrical drawings	A0003	S0013	7.00
D0008	d08_Electrical design report	A0003	S0013	15.00

(8 rows)

DELETE 6

PIDPerson	PersonName	PIDAttribute	PIDStatusElement
P0001	p01_Client	A0003	S0013
P0002	p02_Architect	A0003	S0013
P0003	p03_Structural Engineer	A0003	S0013
P0004	p04_Electrical Engineer	A0003	S0013
P0005	p05_Draftsman	A0003	S0013
P0006	p06_Checking Engineer	A0003	S0013

(6 rows)

DELETE 0

PIDTask	TaskName	LogicalStep	PIDAttribute	PIDStatusElement
T0001	t01_Create architectural design	Step0001	A0003	S0013
T0002	t02_Review architectural design	Step0002	A0003	S0013
T0003	t03_Preliminary structural design	Step0002	A0003	S0013
T0004	t04_Create foundation drawings	Step0003	A0003	S0013
T0005	t05_Create concrete layout drawings	Step0003	A0003	S0013
T0006	t06_Create reinforcement drawings	Step0004	A0003	S0013
T0007	t07_Preliminary electrical design	Step0002	A0003	S0013
T0008	t08_Create electrical drawings	Step0003	A0003	S0013
T0009	t09_Finalize architectural drawings	Step0003	A0003	S0013
T0010	t10_Finalize foundation drawings	Step0004	A0003	S0013
T0011	t11_Finalize concrete layout drawings	Step0004	A0003	S0013
T0012	t12_Finalize reinforcement drawings	Step0005	A0003	S0013
T0013	t13_Finalize electrical drawings	Step0004	A0003	S0013
T0014	t14_Finalize electrical design	Step0003	A0003	S0013
T0015	t15_Finalize structural design	Step0003	A0003	S0013
T0016	t16_Check structural design	Step0004	A0003	S0013
T0017	t17_Check architectural drawings	Step0004	A0003	S0013

T0018	t18_Check foundation drawings	Step0005	A0003	S0013
T0019	t19_Check concrete layout drawings	Step0005	A0003	S0013
T0020	t20_Check reinforcement drawings	Step0006	A0003	S0013
T0021	t21_Check electrical drawings	Step0005	A0003	S0013
T0022	t22_Check electrical design	Step0004	A0003	S0013

(22 rows)

DELETE 0

PIDPerson	PersonName	PIDAttribute	PIDStatusElement
-----------	------------	--------------	------------------

P0001	p01_Client	A0003	S0013
P0002	p02_Architect	A0003	S0013
P0003	p03_Structural Engineer	A0003	S0013
P0004	p04_Electrical Engineer	A0003	S0013
P0005	p05_Draftsman	A0003	S0013
P0006	p06_Checking Engineer	A0003	S0013

(6 rows)

DELETE 0

PIDEdgeDatasetTool	EdgeDatasetToolName	PIDDataset	PIDTool	PIDAttribute	PIDStatusElement
--------------------	---------------------	------------	---------	--------------	------------------

D0001G0002	d01_Client requirements-g02_Word processing software	D0001	G0002	A0003	S0013
D0001G0003	d01_Client requirements-g03_Spreadsheet software	D0001	G0003	A0003	S0013
D0001G0004	d01_Client requirements-g04_Hand calculations	D0001	G0004	A0003	S0013
D0002G0001	d02_Architectural drawings-g01_CAD software	D0002	G0001	A0003	S0013
D0003G0001	d03_Foundation drawings-g01_CAD software	D0003	G0001	A0003	S0013
D0004G0001	d04_Concrete layout drawings-g01_CAD software	D0004	G0001	A0003	S0013
D0005G0001	d05_Reinforcement drawings-g01_CAD software	D0005	G0001	A0003	S0013
D0006G0002	d06_Structural design report-g02_Word processing software	D0006	G0002	A0003	S0013
D0006G0003	d06_Structural design report-g03_Spreadsheet software	D0006	G0003	A0003	S0013
D0006G0004	d06_Structural design report-g04_Hand calculations	D0006	G0004	A0003	S0013
D0007G0001	d07_Electrical drawings-g01_CAD software	D0007	G0001	A0003	S0013
D0008G0002	d08_Electrical design report-g02_Word processing software	D0008	G0002	A0003	S0013
D0008G0003	d08_Electrical design report-g03_Spreadsheet software	D0008	G0003	A0003	S0013

(13 rows)

DELETE 0

PIDEdgeTaskDatasetCreate	EdgeTaskDatasetCreateName	PIDTask	PIDDataset	PIDAttribute	PIDStatusElement
--------------------------	---------------------------	---------	------------	--------------	------------------

T0001D0002	t01_Create architectural design-d02_Architectural drawings	T0001	D0002	A0003	S0001
T0003D0006	t03_Preliminary structural design-d06_Structural design report	T0003	D0006	A0003	S0005
T0004D0003	t04_Create foundation drawings-d03_Foundation drawings	T0004	D0003	A0003	S0005
T0005D0004	t05_Create concrete layout drawings-d04_Concrete layout drawings	T0005	D0004	A0003	S0005
T0006D0005	t06_Create reinforcement drawings-d05_Reinforcement drawings	T0006	D0005	A0003	S0005
T0007D0008	t07_Preliminary electrical design-d08_Electrical design report	T0007	D0008	A0003	S0009
T0008D0007	t08_Create electrical drawings-d07_Electrical drawings	T0008	D0007	A0003	S0005

(7 rows)

DELETE 0

PIDEdgeTaskDatasetModify	EdgeTaskDatasetModifyName	PIDTask	PIDDataset	PIDAttribute	PIDStatusElement
--------------------------	---------------------------	---------	------------	--------------	------------------

T0002D0002	t02_Review architectural design-d02_Architectural drawings	T0002	D0002	A0003	S0002
T0009D0002	t09_Finalize architectural drawings-d02_Architectural drawings	T0009	D0002	A0003	S0003
T0010D0003	t10_Finalize foundation drawings-d03_Foundation drawings	T0010	D0003	A0003	S0007
T0011D0004	t11_Finalize concrete layout drawings-d04_Concrete layout drawings	T0011	D0004	A0003	S0007
T0012D0005	t12_Finalize reinforcement drawings-d05_Reinforcement drawings	T0012	D0005	A0003	S0007
T0013D0007	t13_Finalize electrical drawings-d07_Electrical drawings	T0013	D0007	A0003	S0007
T0014D0008	t14_Finalize electrical design-d08_Electrical design report	T0014	D0008	A0003	S0011
T0015D0006	t15_Finalize structural design-d06_Structural design report	T0015	D0006	A0003	S0011
T0016D0006	t16_Check structural design-d06_Structural design report	T0016	D0006	A0003	S0012
T0017D0002	t17_Check architectural drawings-d02_Architectural drawings	T0017	D0002	A0003	S0004
T0018D0003	t18_Check foundation drawings-d03_Foundation drawings	T0018	D0003	A0003	S0008
T0019D0004	t19_Check concrete layout drawings-d04_Concrete layout drawings	T0019	D0004	A0003	S0008
T0020D0005	t20_Check reinforcement drawings-d05_Reinforcement drawings	T0020	D0005	A0003	S0008
T0021D0007	t21_Check electrical drawings-d07_Electrical drawings	T0021	D0007	A0003	S0008
T0022D0008	t22_Check electrical design-d08_Electrical design report	T0022	D0008	A0003	S0012

(15 rows)

DELETE 0

PIDEdgeTaskDatasetRead	EdgeTaskDatasetReadName	PIDTask	PIDDataset	PIDAttribute	PIDStatusElement
------------------------	-------------------------	---------	------------	--------------	------------------

T0001D0001	t01_Create architectural design-d01_Client requirements	T0001	D0001	A0003	S0013
T0002D0002	t02_Review architectural design-d02_Architectural drawings	T0002	D0002	A0003	S0013
T0003D0002	t03_Preliminary structural design-d02_Architectural drawings	T0003	D0002	A0003	S0013
T0004D0002	t04_Create foundation drawings-d02_Architectural drawings	T0004	D0002	A0003	S0013
T0004D0006	t04_Create foundation drawings-d06_Structural design report	T0004	D0006	A0003	S0013
T0005D0002	t05_Create concrete layout drawings-d02_Architectural drawings	T0005	D0002	A0003	S0013
T0005D0006	t05_Create concrete layout drawings-d06_Structural design report	T0005	D0006	A0003	S0013
T0006D0004	t06_Create reinforcement drawings-d04_Concrete layout drawings	T0006	D0004	A0003	S0013
T0006D0006	t06_Create reinforcement drawings-d06_Structural design report	T0006	D0006	A0003	S0013
T0007D0002	t07_Preliminary electrical design-d02_Architectural drawings	T0007	D0002	A0003	S0013
T0008D0002	t08_Create electrical drawings-d02_Architectural drawings	T0008	D0002	A0003	S0013
T0008D0008	t08_Create electrical drawings-d08_Electrical design report	T0008	D0008	A0003	S0013
T0012D0004	t12_Finalize reinforcement drawings-d04_Concrete layout drawings	T0012	D0004	A0003	S0013

(13 rows)

DELETE 0

PIDEdgeTaskPerson	EdgeTaskPersonName	PIDTask	PIDPerson	PIDAttribute	PIDStatusElement
T0001P0002	t01_Create architectural design-p02_Architect	T0001	P0002	A0003	S0013
T0002P0001	t02_Review architectural design-p01_Client	T0002	P0001	A0003	S0013
T0002P0002	t02_Review architectural design-p02_Architect	T0002	P0002	A0003	S0013
T0003P0003	t03_Preliminary structural design-p03_Structural Engineer	T0003	P0003	A0003	S0013
T0005P0005	t05_Create concrete layout drawings-p05_Draftsman	T0005	P0005	A0003	S0013
T0006P0005	t06_Create reinforcement drawings-p05_Draftsman	T0006	P0005	A0003	S0013
T0007P0005	t07_Preliminary electrical design-p05_Draftsman	T0007	P0005	A0003	S0013
T0008P0005	t08_Create electrical drawings-p05_Draftsman	T0008	P0005	A0003	S0013
T0009P0002	t09_Finalize architectural drawings-p02_Architect	T0009	P0002	A0003	S0013
T0009P0005	t09_Finalize architectural drawings-p05_Draftsman	T0009	P0005	A0003	S0013
T0010P0003	t10_Finalize foundation drawings-p03_Structural Engineer	T0010	P0003	A0003	S0013
T0010P0005	t10_Finalize foundation drawings-p05_Draftsman	T0010	P0005	A0003	S0013
T0011P0002	t11_Finalize concrete layout drawings-p02_Architect	T0011	P0002	A0003	S0013
T0011P0003	t11_Finalize concrete layout drawings-p03_Structural Engineer	T0011	P0003	A0003	S0013
T0011P0005	t11_Finalize concrete layout drawings-p05_Draftsman	T0011	P0005	A0003	S0013
T0012P0003	t12_Finalize reinforcement drawings-p03_Structural Engineer	T0012	P0003	A0003	S0013
T0012P0005	t12_Finalize reinforcement drawings-p05_Draftsman	T0012	P0005	A0003	S0013
T0013P0004	t13_Finalize electrical drawings-p04_Electrical Engineer	T0013	P0004	A0003	S0013
T0013P0005	t13_Finalize electrical drawings-p05_Draftsman	T0013	P0005	A0003	S0013
T0014P0004	t14_Finalize electrical design-p04_Electrical Engineer	T0014	P0004	A0003	S0013
T0015P0003	t15_Finalize structural design-p03_Structural Engineer	T0015	P0003	A0003	S0013
T0016P0006	t16_Check structural design-p06_Checking Engineer	T0016	P0006	A0003	S0013
T0017P0006	t17_Check architectural drawings-p06_Checking Engineer	T0017	P0006	A0003	S0013
T0018P0006	t18_Check foundation drawings-p06_Checking Engineer	T0018	P0006	A0003	S0013
T0019P0006	t19_Check concrete layout drawings-p06_Checking Engineer	T0019	P0006	A0003	S0013
T0020P0006	t20_Check reinforcement drawings-p06_Checking Engineer	T0020	P0006	A0003	S0013
T0021P0006	t21_Check electrical drawings-p06_Checking Engineer	T0021	P0006	A0003	S0013
T0022P0006	t22_Check electrical design-p06_Checking Engineer	T0022	P0006	A0003	S0013

(28 rows)

DELETE 0

PIDEdgeTaskTask	EdgeTaskTaskName	PIDTaskIn	PIDTaskOut	PIDAttribute	PIDStatusElement	PIDTask
T0001T0002	t01_Create architectural design-t02_Review architectural design	T0001	T0002	A0003	S0013	T0001
T0001T0003	t01_Create architectural design-t03_Preliminary structural design	T0001	T0003	A0003	S0013	T0001
T0001T0004	t01_Create architectural design-t04_Create foundation drawings	T0001	T0004	A0003	S0013	T0001
T0001T0005	t01_Create architectural design-t05_Create concrete layout drawings	T0001	T0005	A0003	S0013	T0001
T0001T0007	t01_Create architectural design-t07_Preliminary electrical design	T0001	T0007	A0003	S0013	T0001
T0001T0008	t01_Create architectural design-t08_Create electrical drawings	T0001	T0008	A0003	S0013	T0001
T0001T0009	t01_Create architectural design-t09_Finalize architectural drawings	T0001	T0009	A0003	S0013	T0001
T0001T0017	t01_Create architectural design-t17_Check architectural drawings	T0001	T0017	A0003	S0013	T0001
T0002T0009	t02_Review architectural design-t09_Finalize architectural drawings	T0002	T0009	A0003	S0013	T0002
T0002T0017	t02_Review architectural design-t17_Check architectural drawings	T0002	T0017	A0003	S0013	T0002
T0003T0004	t03_Preliminary structural design-t04_Create foundation drawings	T0003	T0004	A0003	S0013	T0003
T0003T0005	t03_Preliminary structural design-t05_Create concrete layout drawings	T0003	T0005	A0003	S0013	T0003
T0003T0006	t03_Preliminary structural design-t06_Create reinforcement drawings	T0003	T0006	A0003	S0013	T0003
T0003T0015	t03_Preliminary structural design-t15_Finalize structural design	T0003	T0015	A0003	S0013	T0003
T0003T0016	t03_Preliminary structural design-t16_Check structural design	T0003	T0016	A0003	S0013	T0003
T0004T0010	t04_Create foundation drawings-t10_Finalize foundation drawings	T0004	T0010	A0003	S0013	T0004
T0004T0018	t04_Create foundation drawings-t18_Check foundation drawings	T0004	T0018	A0003	S0013	T0004
T0005T0006	t05_Create concrete layout drawings-t06_Create reinforcement drawings	T0005	T0006	A0003	S0013	T0005
T0005T0011	t05_Create concrete layout drawings-t11_Finalize concrete layout drawings	T0005	T0011	A0003	S0013	T0005
T0005T0012	t05_Create concrete layout drawings-t12_Finalize reinforcement drawings	T0005	T0012	A0003	S0013	T0005
T0005T0019	t05_Create concrete layout drawings-t19_Check concrete layout drawings	T0005	T0019	A0003	S0013	T0005
T0006T0012	t06_Create reinforcement drawings-t12_Finalize reinforcement drawings	T0006	T0012	A0003	S0013	T0006
T0006T0020	t06_Create reinforcement drawings-t20_Check reinforcement drawings	T0006	T0020	A0003	S0013	T0006
T0007T0008	t07_Preliminary electrical design-t08_Create electrical drawings	T0007	T0008	A0003	S0013	T0007
T0007T0014	t07_Preliminary electrical design-t14_Finalize electrical design	T0007	T0014	A0003	S0013	T0007
T0007T0022	t07_Preliminary electrical design-t22_Check electrical design	T0007	T0022	A0003	S0013	T0007
T0008T0013	t08_Create electrical drawings-t13_Finalize electrical drawings	T0008	T0013	A0003	S0013	T0008
T0008T0021	t08_Create electrical drawings-t21_Check electrical drawings	T0008	T0021	A0003	S0013	T0008
T0009T0017	t09_Finalize architectural drawings-t17_Check architectural drawings	T0009	T0017	A0003	S0013	T0009
T0010T0018	t10_Finalize foundation drawings-t18_Check foundation drawings	T0010	T0018	A0003	S0013	T0010
T0011T0019	t11_Finalize concrete layout drawings-t19_Check concrete layout drawings	T0011	T0019	A0003	S0013	T0011
T0012T0020	t12_Finalize reinforcement drawings-t20_Check reinforcement drawings	T0012	T0020	A0003	S0013	T0012
T0013T0021	t13_Finalize electrical drawings-t21_Check electrical drawings	T0013	T0021	A0003	S0013	T0013
T0014T0022	t14_Finalize electrical design-t22_Check electrical design	T0014	T0022	A0003	S0013	T0014
T0015T0016	t15_Finalize structural design-t16_Check structural design	T0015	T0016	A0003	S0013	T0015

(35 rows)

DELETE 0

PIDEdgeTaskTool	EdgeTaskToolName	PIDTask	PIDTool	PIDAttribute	PIDStatusElement
T0001G0002	t01_Create architectural design-g02_Word processing software	T0001	G0002	A0003	S0013
T0001G0003	t01_Create architectural design-g03_Spreadsheet software	T0001	G0003	A0003	S0013
T0001G0004	t01_Create architectural design-g04_Hand calculations	T0001	G0004	A0003	S0013
T0002G0001	t02_Review architectural design-g01_CAD software	T0002	G0001	A0003	S0013
T0003G0001	t03_Preliminary structural design-g01_CAD software	T0003	G0001	A0003	S0013
T0004G0001	t04_Create foundation drawings-g01_CAD software	T0004	G0001	A0003	S0013
T0004G0002	t04_Create foundation drawings-g02_Word processing software	T0004	G0002	A0003	S0013
T0004G0003	t04_Create foundation drawings-g03_Spreadsheet software	T0004	G0003	A0003	S0013
T0004G0004	t04_Create foundation drawings-g04_Hand calculations	T0004	G0004	A0003	S0013
T0005G0001	t05_Create concrete layout drawings-g01_CAD software	T0005	G0001	A0003	S0013
T0005G0002	t05_Create concrete layout drawings-g02_Word processing software	T0005	G0002	A0003	S0013
T0005G0003	t05_Create concrete layout drawings-g03_Spreadsheet software	T0005	G0003	A0003	S0013
T0005G0004	t05_Create concrete layout drawings-g04_Hand calculations	T0005	G0004	A0003	S0013
T0006G0001	t06_Create reinforcement drawings-g01_CAD software	T0006	G0001	A0003	S0013
T0006G0002	t06_Create reinforcement drawings-g02_Word processing software	T0006	G0002	A0003	S0013
T0006G0003	t06_Create reinforcement drawings-g03_Spreadsheet software	T0006	G0003	A0003	S0013
T0006G0004	t06_Create reinforcement drawings-g04_Hand calculations	T0006	G0004	A0003	S0013

T0007G0001	t07_Preliminary electrical design-g01_CAD software	T0007	G0001	A0003	S0013
T0008G0001	t08_Create electrical drawings-g01_CAD software	T0008	G0001	A0003	S0013
T0008G0002	t08_Create electrical drawings-g02_Word processing software	T0008	G0002	A0003	S0013
T0008G0003	t08_Create electrical drawings-g03_Spreadsheet software	T0008	G0003	A0003	S0013
T0012G0001	t12_Finalize reinforcement drawings-g01_CAD software	T0012	G0001	A0003	S0013
(22 rows)					

J.10 Importing database data using the Java JDBC-ODBC bridge

Java JDBC-ODBC bridge is used here for data loading. This is an alternative approach to populating database directly from Engineering process modelling application

```
package JDBCODBCDatabase;
import java.io.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.*;
public class DatabaseDataInput {
    static String [] Files;
    static String [] Tables;
    static final String DRIVER_NAME = "sun.jdbc.odbc.JdbcOdbcDriver";
    // Database name hardcoded
    static final String DATABASE_URL = "jdbc:odbc:PostgreSQLjavbs";
    /**
     * @param args
     */
    public static void main(String[] args) throws IOException
    {
        // TODO Auto-generated method stub
        // Set up input file and table references
        Files=fileNames();
        Tables=tableNames();
        for (int ifiles=0;ifiles<Files.length;ifiles++)
        {
            System.out.println(ifiles);
            // Check file existence
            File csvFile= new File(Files[ifiles]);
            if(csvFile.canRead())
            {
                System.out.println(Files [ifiles]+" can be read");
            }
            else
            {
                System.out.println(Files [ifiles]+" can not be read");
            }
            //Open file
            FileReader iStream= new FileReader(Files[ifiles]);
            // Reader iStream= new Reader(csvFile);
            BufferedReader bRead= new BufferedReader(iStream);
            //Read records and store in appropriate table

            try{
                String record;
                while((record = bRead.readLine())!= null)
                {
                    StringTokenizer stk= new StringTokenizer (record , "," );
                    String SQLData="";
                    while (stk.hasMoreTokens())
                    {
                        SQLData=SQLData+" '"+stk.nextToken()+"',";
                    }
                    // remove last , from string
                    SQLData=SQLData.substring(0,SQLData.length()-1);
                    databaseDataPut(SQLData,Tables[ifiles]);
                }
            }
            catch (IOException iox)
            {
                System.out.println( iox);
            }
            finally
            {
                bRead.close();
                System.out.println(Files [ifiles]+" data inserted");
            }
            // end of finally
        }
        // end of for loop
    }
    // end of method

    public static int databaseDataPut(String Record, String Table)
    {
        int nrows=0;
        try {
            Class.forName(DRIVER_NAME);
            Connection connection = null;
            connection = DriverManager.getConnection(DATABASE_URL);
            Statement statement=connection.createStatement();
            // "INSERT INTO \"tbAttribute\" VALUES ('xx');"
            String SQLStatement="INSERT INTO "+Table
            +" VALUES("+ Record+" )";
        }
    }
}
```

```

        System.out.println(SQLStatement);

        nrows=statement.executeUpdate(SQLStatement);
    }
    catch (ClassNotFoundException cnfe)
    {
        System.err.println("ClassNotFoundException Was Thrown");
        cnfe.printStackTrace();
    }
    catch (SQLException sqle)
    {
        System.err.println("SQLException Was Thrown");
        sqle.printStackTrace();
    }
    return nrows;
}

public static String [] fileNames()
{
    // Sequence of file names must correspond to
    // sequence of loading of tables is important for foreign key referencing
    Files = new String [13];
    Files[0]="PEPEExtbAttribute.csv";
    Files[1]="PEPEExtbDataset.csv";
    //
    Files[2]="PEPEExtbPerson.csv";
    Files[3]="PEPEExtbStatusElement.csv";
    Files[4]="PEPEExtbTask.csv";
    Files[5]="PEPEExtbTool.csv";
    //
    Files[6]="PEPEExtbEdgeDatasetTool.csv";
    //
    Files[7]="PEPEExtbEdgeTaskDataset.csv"; // Not used see R C W
    Files[7]="PEPEExtbEdgeTaskDatasetCreate.csv";
    Files[8]="PEPEExtbEdgeTaskDatasetModify.csv";
    Files[9]="PEPEExtbEdgeTaskDatasetRead.csv";
    Files[10]="PEPEExtbEdgeTaskPerson.csv";
    Files[11]="PEPEExtbEdgeTaskTask.csv";
    Files[12]="PEPEExtbEdgeTaskTool.csv";
    return Files;
}

public static String [] tableNames()
{
    // Sequence of loading of tables is important for foreign key referencing
    Tables = new String [13];
    Tables[0]=" \"tbAttribute\" ";
    Tables[1]=" \"tbDataset\" ";
    //
    Tables[2]=" \"tbPerson\" ";
    Tables[3]=" \"tbStatusElement\" ";
    Tables[4]=" \"tbTask\" ";
    Tables[5]=" \"tbTool\" ";
    //
    Tables[6]=" \"tbEdgeDatasetTool\" ";
    //
    Tables[7]=" \"tbEdgeTaskDataset\" "; // Not used see R C W
    Tables[7]=" \"tbEdgeTaskDatasetCreate\" ";
    Tables[8]=" \"tbEdgeTaskDatasetModify\" ";
    Tables[9]=" \"tbEdgeTaskDatasetRead\" ";
    Tables[10]=" \"tbEdgeTaskPerson\" ";
    Tables[11]=" \"tbEdgeTaskTask\" ";
    Tables[12]=" \"tbEdgeTaskTool\" ";
    return Tables;
}
}

```

J.11 Database application SQL functionality availability and usage

Microsoft Access SQL union queries

Action queries, as well as SQL UNION queries, cannot be used as a row source i.e. saved as a table in Access. One needs to first create a union query, and then use the results of that query in a Access make-table query. Refer to <http://support.microsoft.com/default.aspx?scid=kb;en-us;208819> quoted below.

Article ID : 208819 Last Review : July 13, 2004 Revision : 1.0 This article was previously published under Q208819 Moderate: Requires basic macro, coding, and interoperability skills.

This article applies only to a Microsoft Access database (.mdb).

SUMMARY

Microsoft Access SQL does not allow you to use the INTO clause (a clause needed to create a make-table query) within a union query. Therefore, you cannot directly create a make-table query; you must first create a union query, and then use the results of that query in the make-table query. This article demonstrates how to do this.

NOTE: You can see a demonstration of the technique that is used in this article in the sample file Qrysmpl00.exe. For information about how to obtain this sample file, please see the following article in the Microsoft Knowledge Base: 207626 (<http://support.microsoft.com/kb/207626/EN-US/>) ACC2000: Access 2000 Sample Queries Available in Download Center

MORE INFORMATION

To create a table from a union query, you must first define the union query, and then create a make-table query based on the union query results. To do so, follow these steps:

CAUTION: If you follow the steps in this example, you modify the sample database Northwind.mdb. You may want to back up the Northwind.mdb file and follow these steps on a copy of the database.

1. Start Microsoft Access, and then open the sample database Northwind.mdb.
2. Create a new query. In the New Query dialog box, click Design View, and then click OK.
3. Close the Show Table dialog box. On the Query menu, point to SQL Specific, and then click Union.
4. Type the following lines into the SQL window:

```
SELECT CompanyName, City, "Customers" as [Relationship] FROM Customers
WHERE Country = "Brazil" UNION SELECT CompanyName, City, "Suppliers"
FROM Suppliers WHERE Country = "Brazil";
```

5. Save the query as qryMyUnion, and then close the SQL window.
6. Save the query as qryMyUnion, and then close the SQL window.
7. Create a new query based on qryMyUnion, and then close the Show Tables dialog box.
8. Double-click the qryMyUnion query's asterisk (*) to add all the fields to the query's output. On the Query menu, click Make Table. In the Table Name box, type tblMyUnion, and then click OK.
9. On the Query menu, click Run, and then click Yes on the dialog box that informs you how many records will be copied into the new table.
10. Save the query as qryMyUnionMakeTable, and then close the query.
11. Open table tblMyUnion. Note that the query qryMyUnionMakeTable created 10 records from the Customers and Suppliers tables whose Country field contained "Brazil."

REFERENCES

For more information about union queries, click Microsoft Access Help on the Help menu, type what is an sql query and when would you use one in the Office Assistant or the Answer Wizard, and then click Search to view the topic.

For more information about make-table queries, click Microsoft Access Help on the Help menu, type create a new table from the results of a query with a make-table query in the Office Assistant or the Answer Wizard, and then click Search to view the topic.

J.11.1 SQL Queries for S-Curve presentation

Desktop standalone database application with data file imports

First step query for S-Curve recordset generation. File: *SCurveStep1Local.sql*

```

SELECT

[tbTask].[LogicalStep]

, [tbEdgeTaskDatasetCreate].[EdgeTaskDatasetCreateName]
, [tbEdgeTaskDatasetCreate].[PIDDataset] AS [PIDDatasetCreate]
, [tbEdgeTaskDatasetCreate].[PIDStatusElement] AS [PIDStatusCreate]

, [tbEdgeTaskDatasetRead].[EdgeTaskDatasetReadName]
, [tbEdgeTaskDatasetRead].[PIDDataset] AS [PIDDatasetRead]
, [tbEdgeTaskDatasetRead].[PIDStatusElement] AS [PIDStatusRead]

, [tbEdgeTaskDatasetModify].[EdgeTaskDatasetModifyName]
, [tbEdgeTaskDatasetModify].[PIDDataset] AS [PIDDatasetModify]
, [tbEdgeTaskDatasetModify].[PIDStatusElement] AS [PIDStatusModify]

INTO [tbSCurveStep1]

FROM

[tbEdgeTaskDatasetRead]

RIGHT JOIN ([tbEdgeTaskDatasetModify]
RIGHT JOIN ([tbEdgeTaskDatasetCreate]
RIGHT JOIN [tbTask]
ON [tbEdgeTaskDatasetCreate].[PIDTask] = [tbTask].[PIDTask])
ON [tbEdgeTaskDatasetModify].[PIDTask] = [tbTask].[PIDTask])
ON [tbEdgeTaskDatasetRead].[PIDTask] = [tbTask].[PIDTask]

ORDER BY [tbTask].[LogicalStep]
;
\end{{boxedverbatim}}
\normalsize

Second step query for S-Curve recordset generation.
File: \emph{SCurveStep2.sql}
\scriptsize
\begin{{boxedverbatim}}
SELECT DISTINCT

tbSCurveStep1.LogicalStep AS LogicalStep

, public_tbDataset.PIDDataset AS PIDDataset
, public_tbDataset.DatasetName AS DatasetName

, public_tbDataset.DatasetName AS DatasetCreateName
, public_tbDataset.Weight AS WeightCreate
, public_tbStatusElement.Completion AS CompletionCreate
, public_tbDataset.Weight*
public_tbStatusElement.Completion
/100 AS CompletionCreateWeight

, NULL AS DatasetReadName
, 0.0 AS WeightRead
, 0.0 AS CompletionRead
, 0.0 AS CompletionReadWeight

, NULL AS DatasetModifyName
, 0.0 AS WeightModify
, 0.0 AS CompletionModify
, 0.0 AS CompletionModifyWeight

FROM

(
(tbSCurveStep1
LEFT JOIN public_tbDataset
ON tbSCurveStep1.PIDDatasetCreate = public_tbDataset.PIDDataset)
LEFT JOIN public_tbStatusElement
ON tbSCurveStep1.PIDStatusCreate = public_tbStatusElement.PIDStatusElement
)

UNION

SELECT DISTINCT

```

```

    tbSCurveStep1.LogicalStep AS LogicalStep

, public_tbDataset.PIDDataset AS PIDDataset
, public_tbDataset.DatasetName AS DatasetName

, NULL AS DatasetCreateName
, 0.0 AS WeightCreate
, 0.0 AS CompletionCreate
, 0.0 AS CompletionCreateWeight

, public_tbDataset.DatasetName AS DatasetReadName
, public_tbDataset.Weight AS WeightRead
, public_tbStatusElement.Completion AS CompletionRead
, public_tbDataset.Weight*
    public_tbStatusElement.Completion
    /100 AS CompletionReadWeight

, NULL AS DatasetModifyName
, 0.0 AS WeightModify
, 0.0 AS CompletionModify
, 0.0 AS CompletionModifyWeight

FROM

(
    tbSCurveStep1
LEFT JOIN public_tbDataset
    ON tbSCurveStep1.PIDDatasetRead = public_tbDataset.PIDDataset)
LEFT JOIN public_tbStatusElement
    ON tbSCurveStep1.PIDStatusRead = public_tbStatusElement.PIDStatusElement

UNION

SELECT DISTINCT

    tbSCurveStep1.LogicalStep AS LogicalStep

, public_tbDataset.PIDDataset AS PIDDataset
, public_tbDataset.DatasetName AS DatasetName

, NULL AS DatasetCreateName
, 0.0 AS WeightCreate
, 0.0 AS CompletionCreate
, 0.0 AS CompletionCreateWeight

, NULL AS DatasetReadName
, 0.0 AS WeightRead
, 0.0 AS CompletionRead
, 0.0 AS CompletionReadWeight

, public_tbDataset.DatasetName AS DatasetModifyName
, public_tbDataset.Weight AS WeightModify
, public_tbStatusElement.Completion AS CompletionModify
, public_tbDataset.Weight*
    public_tbStatusElement.Completion
    /100.0 AS CompletionModifyWeight

FROM

(
    (tbSCurveStep1
LEFT JOIN public_tbDataset
    ON tbSCurveStep1.PIDDatasetModify = public_tbDataset.PIDDataset)
LEFT JOIN public_tbStatusElement
    ON tbSCurveStep1.PIDStatusModify = public_tbStatusElement.PIDStatusElement
    )
ORDER BY LogicalStep
;

```

Third query for S-Curve Access table generation. File: *qryTbSCurveStep2.sql*

```

SELECT SCurveStep2All.* INTO tbSCurveStep2
FROM SCurveStep2All;

```

Fourth query for S-Curve percentage summary table generation. File: *SCurveStepPctSummary.sql*

```

SCurveStepPctSummary.sql

SELECT tb.LogicalStep
, (

```



```

SELECT
ROUND(
SUM(CompletionCreateWeight)
,2)
FROM tbSCurveStep2
WHERE
tbSCurveStep2.LogicalStep = tb.LogicalStep
)
AS [RunTotWghtCreate]

,ROUND(
100*(SELECT SUM(CompletionCreateWeight)
/
(SELECT(SUM(WeightCreate)) FROM tbSCurveStep2)
FROM tbSCurveStep2
WHERE
tbSCurveStep2.LogicalStep = tb.LogicalStep
)
,2)
AS [%RunTotWghtCreate]

,(
SELECT
ROUND(
SUM(CompletionModifyWeight)
,2)
FROM tbSCurveStep2
WHERE
tbSCurveStep2.LogicalStep = tb.LogicalStep
)
AS [RunTotWghtModify]

, ROUND(
100*(SELECT SUM(CompletionModifyWeight)
/
(SELECT(SUM(WeightCreate)) FROM tbSCurveStep2)
FROM tbSCurveStep2
WHERE
tbSCurveStep2.LogicalStep = tb.LogicalStep
)
,2)
AS [%RunTotWghtModify]

, ROUND(
100*(SELECT(
SUM(CompletionCreateWeight)
+ SUM(CompletionReadWeight)
+ SUM(CompletionModifyWeight)
)
/
(SELECT(SUM(WeightCreate)) FROM tbSCurveStep2)
FROM tbSCurveStep2
WHERE
tbSCurveStep2.LogicalStep = tb.LogicalStep
)
,2)
AS [%RunTotWght]

, ROUND(
100*(SELECT(
SUM(CompletionCreateWeight)
+ SUM(CompletionReadWeight)
+ SUM(CompletionModifyWeight)
)
/
(SELECT(SUM(WeightCreate)) FROM tbSCurveStep2)
FROM tbSCurveStep2
WHERE
tbSCurveStep2.LogicalStep <= tb.LogicalStep
)
,2)
AS [%TotWght]

```

INTO tbSCurveStepPctSummary

```

FROM tbSCurveStep2 AS tb

GROUP BY tb.LogicalStep
ORDER BY tb.LogicalStep ASC
;

```

Desktop client with database server application

First step query for S-Curve data recordset generation. File: *SCurveStep1.sql*

```

SCurveStep1.sql
SELECT

[public_tbTask].[LogicalStep]

, [public_tbEdgeTaskDatasetCreate].[EdgeTaskDatasetCreateName]
, [public_tbEdgeTaskDatasetCreate].[PIDDataset] AS [PIDDatasetCreate]
, [public_tbEdgeTaskDatasetCreate].[PIDStatusElement] AS [PIDStatusCreate]

, [public_tbEdgeTaskDatasetRead].[EdgeTaskDatasetReadName]
, [public_tbEdgeTaskDatasetRead].[PIDDataset] AS [PIDDatasetRead]
, [public_tbEdgeTaskDatasetRead].[PIDStatusElement] AS [PIDStatusRead]

, [public_tbEdgeTaskDatasetModify].[EdgeTaskDatasetModifyName]
, [public_tbEdgeTaskDatasetModify].[PIDDataset] AS [PIDDatasetModify]
, [public_tbEdgeTaskDatasetModify].[PIDStatusElement] AS [PIDStatusModify]

INTO [tbSCurveStep1]

FROM

[public_tbEdgeTaskDatasetRead]

RIGHT JOIN ([public_tbEdgeTaskDatasetModify]
RIGHT JOIN ([public_tbEdgeTaskDatasetCreate]
RIGHT JOIN [public_tbTask]
ON [public_tbEdgeTaskDatasetCreate].[PIDTask] = [public_tbTask].[PIDTask])
ON [public_tbEdgeTaskDatasetModify].[PIDTask] = [public_tbTask].[PIDTask])
ON [public_tbEdgeTaskDatasetRead].[PIDTask] = [public_tbTask].[PIDTask]

ORDER BY [public_tbTask].[LogicalStep]
;
\end{boxedverbatim}
\normalsize

```

The second, third and fourth step are as previous.

```

\subsubsection {Access Table data output in .csv format}
\scriptsize
\begin{boxedverbatim}
tbSCurveStepPctSummary
LogicalStep      RunTotWghtCreate
%RunTotWghtCreate      RunTotWghtModify
%RunTotWghtModify      %RunTotWght      %TotWght

Step0001          5          3          0          0          3.47          3.47
Step0002          8          5          5          3.47          9.03          12.5
Step0003         13.4          9         40.5         28.12         37.43         49.93
Step0004          2.4          1         56.75         39.41         41.08         91.01
Step0005           0          0         12.35          8.58          8.58         99.58
Step0006           0          0          0.6          0.42          0.42         100

```

Direct database server application

First step query for S-Curve recordset generation. File: *SCurveStep1.sql*

```

DROP TABLE "tbSCurveStep1";

SELECT
"tbTask"."LogicalStep"
-- , *
, "tbEdgeTaskDatasetCreate"."EdgeTaskDatasetCreateName"
, "tbEdgeTaskDatasetCreate"."PIDDataset" AS "PIDDatasetCreate"
, "tbEdgeTaskDatasetCreate"."PIDStatusElement" AS "PIDStatusCreate"

```

```

, "tbEdgeTaskDatasetRead"."EdgeTaskDatasetReadName"
, "tbEdgeTaskDatasetRead"."PIDDataset" AS "PIDDatasetRead"
, "tbEdgeTaskDatasetRead"."PIDStatusElement" AS "PIDStatusRead"

, "tbEdgeTaskDatasetModify"."EdgeTaskDatasetModifyName"
, "tbEdgeTaskDatasetModify"."PIDDataset" AS "PIDDatasetModify"
, "tbEdgeTaskDatasetModify"."PIDStatusElement" AS "PIDStatusModify"

-- Access
-- INTO "tbSCurveStep1"
-- PSQL
INTO TABLE "tbSCurveStep1"

FROM

"tbEdgeTaskDatasetRead"
RIGHT JOIN ("tbEdgeTaskDatasetModify"
RIGHT JOIN ("tbEdgeTaskDatasetCreate"
RIGHT JOIN "tbTask"
ON "tbEdgeTaskDatasetCreate"."PIDTask" = "tbTask"."PIDTask")
ON "tbEdgeTaskDatasetModify"."PIDTask" = "tbTask"."PIDTask")
ON "tbEdgeTaskDatasetRead"."PIDTask" = "tbTask"."PIDTask"

ORDER BY "tbTask"."LogicalStep"
;

SELECT * FROM "tbSCurveStep1";
\end{{boxedverbatim}}
\normalsize

Second step query for S-Curve data table generation.
File: \emph{SCurveStep1.sql}
\scriptsize
\begin{{boxedverbatim}}
DROP TABLE "tbSCurveStep2";

SELECT DISTINCT

    "tbSCurveStep1"."LogicalStep" AS "LogicalStep"
, "tbDataset"."PIDDataset" AS "PIDDataset"
, "tbDataset"."DatasetName" AS "DatasetName"
-- Create Columns
, "tbDataset"."DatasetName" AS "DatasetCreateName"
, "tbDataset"."Weight" AS "WeightCreate"
, "tbStatusElement"."Completion" AS "CompletionCreate"
, ROUND(
    "tbDataset"."Weight"*
    "tbStatusElement"."Completion"
    /100
,2)
    AS "CompletionCreateWeight"
-- Read Columns
, NULL AS "DatasetReadName"
, 0.0 AS "WeightRead"
, 0.0 AS "CompletionRead"
, 0.0 AS "CompletionReadWeight"
-- Modify Columns
, NULL AS "DatasetModifyName"
, 0.0 AS "WeightModify"
, 0.0 AS "CompletionModify"
, 0.0 AS "CompletionModifyWeight"

-- Only on first select
INTO TABLE "tbSCurveStep2"

FROM
-- Create Tables
(
("tbSCurveStep1"
LEFT JOIN "tbDataset"
ON "tbSCurveStep1"."PIDDatasetCreate" = "tbDataset"."PIDDataset")
LEFT JOIN "tbStatusElement"
ON "tbSCurveStep1"."PIDStatusCreate" = "tbStatusElement"."PIDStatusElement"
)
-- ORDER BY "tbSCurveStep1"."LogicalStep"

UNION

SELECT DISTINCT

    "tbSCurveStep1"."LogicalStep" AS "LogicalStep"
, "tbDataset"."PIDDataset" AS "PIDDataset"
, "tbDataset"."DatasetName" AS "DatasetName"
-- Create Columns
, NULL AS "DatasetCreateName"

```

```

, 0.0 AS "WeightCreate"
, 0.0 AS "CompletionCreate"
, 0.0 AS "CompletionCreateWeight"
-- Read Columns
, "tbDataset"."DatasetName" AS "DatasetReadName"
, "tbDataset"."Weight" AS "WeightRead"
, "tbStatusElement"."Completion" AS "CompletionRead"
, ROUND(
    "tbDataset"."Weight"*
    "tbStatusElement"."Completion"
    /100
,2)
    AS "CompletionReadWeight"
-- Modify Columns
, NULL AS "DatasetModifyName"
, 0.0 AS "WeightModify"
, 0.0 AS "CompletionModify"
, 0.0 AS "CompletionModifyWeight"

-- Only on first select
-- INTO TABLE "tbSCurveStep2"

FROM
-- Read tables
(
    "tbSCurveStep1"
LEFT JOIN "tbDataset"
    ON "tbSCurveStep1"."PIDDatasetRead" = "tbDataset"."PIDDataset")
LEFT JOIN "tbStatusElement"
    ON "tbSCurveStep1"."PIDStatusRead" = "tbStatusElement"."PIDStatusElement"

UNION

SELECT DISTINCT

    "tbSCurveStep1"."LogicalStep" AS "LogicalStep"
, "tbDataset"."PIDDataset" AS "PIDDataset"
, "tbDataset"."DatasetName" AS "DatasetName"
-- Create Columns
, NULL AS "DatasetCreateName"
, 0.0 AS "WeightCreate"
, 0.0 AS "CompletionCreate"
, 0.0 AS "CompletionCreateWeight"
-- Read Columns
, NULL AS "DatasetReadName"
, 0.0 AS "WeightRead"
, 0.0 AS "CompletionRead"
, 0.0 AS "CompletionReadWeight"
-- Modify Columns
, "tbDataset"."DatasetName" AS "DatasetModifyName"
, "tbDataset"."Weight" AS "WeightModify"
, "tbStatusElement"."Completion" AS "CompletionModify"
, ROUND(
    "tbDataset"."Weight"*
    "tbStatusElement"."Completion"
    /100.0
,2)
    AS "CompletionModifyWeight"

-- Only on first select
-- INTO TABLE "tbSCurveStep2"

FROM
-- Modify Tables
(
    ("tbSCurveStep1"
LEFT JOIN "tbDataset"
    ON "tbSCurveStep1"."PIDDatasetModify" = "tbDataset"."PIDDataset")
LEFT JOIN "tbStatusElement"
    ON "tbSCurveStep1"."PIDStatusModify" = "tbStatusElement"."PIDStatusElement"
    )

ORDER BY "LogicalStep"
;

-- Remove null entries - Developed in Union process

DELETE FROM "tbSCurveStep2"
    WHERE "PIDDataset" ISNULL;

SELECT * FROM "tbSCurveStep2";

-- PSQL COPY only for superusers use \copy
-- COPY "tbSCurveStep2" TO 'tbSCurveStep2.csv' DELIMITER ',';
\!rm tbSCurveStep2.csv

```

```

\copy "tbSCurveStep2" TO 'tbSCurveStep2.csv' DELIMITER ','

\end{{boxedverbatim}}
\normalsize

Third step query for S-Curve percentage summary data table generation.
File: \emph{SCurveStepPctSummary.sql}
\scriptsize
\begin{{boxedverbatim}}
SCurveStepPctSummary.sql

SELECT version();
SELECT current_date;
DROP TABLE "tbSCurvePercentage";

--          Table "public.tbSCurveStep2"
--          Column          |          Type
-----+-----
-- LogicalStep             | character(10)
-- PIDDataset              | character(10)
-- DatasetName              | character varying(100)
-- DatasetCreateName       | character varying
-- WeightCreate             | numeric
-- CompletionCreate         | numeric
-- CompletionCreateWeight   | numeric
-- DatasetReadName         | text
-- WeightRead              | numeric
-- CompletionRead           | numeric
-- CompletionReadWeight     | numeric
-- DatasetModifyName       | text
-- WeightModify            | numeric
-- CompletionModify        | numeric
-- CompletionModifyWeight   | numeric

SELECT tb."LogicalStep"
      ,tb."PIDDataset"
      ,tb."DatasetName"

-- Data create values and percentages

      ,tb."WeightCreate"
      ,ROUND(tb."CompletionCreate",2) AS "CompletionCreate"
      ,ROUND(tb."CompletionCreateWeight",2) AS "CompletionCreateWeight"

      ,(
        SELECT
        ROUND(
        SUM("CompletionCreateWeight")
        ,2)
        FROM "tbSCurveStep2"
        WHERE "tbSCurveStep2"."PIDDataset" <= tb."PIDDataset")
        AS "Running Total Weight Create"
      ,ROUND(
        100*(SELECT SUM("CompletionCreateWeight")
        /
        (SELECT(SUM("WeightCreate")) FROM "tbSCurveStep2")
        FROM "tbSCurveStep2"
        WHERE "tbSCurveStep2"."PIDDataset" <= tb."PIDDataset"
        AND
        "tbSCurveStep2"."LogicalStep" = tb."LogicalStep"
        )
        ,2)
        AS "Percent Running Total Weight Create"

-- Data read values and percentages
-- WeightCreate is assumed to sum to the total weight

      ,tb."WeightRead"
      ,ROUND(tb."CompletionRead",2) AS "CompletionRead"
      ,ROUND(tb."CompletionReadWeight",2) AS "CompletionReadWeight"

      ,(
        SELECT
        ROUND(
        SUM("CompletionReadWeight")
        ,2)
        FROM "tbSCurveStep2"
        WHERE "tbSCurveStep2"."PIDDataset" <= tb."PIDDataset")
        AS "Running Total Weight Read"
      ,ROUND(
        100*(SELECT SUM("CompletionReadWeight")
        /
        (SELECT(SUM("WeightCreate")) FROM "tbSCurveStep2")

```

```

        FROM "tbSCurveStep2"
        WHERE "tbSCurveStep2"."PIDDataset" <= tb."PIDDataset"
        AND
        "tbSCurveStep2"."LogicalStep" = tb."LogicalStep"
    )
    ,2)
    AS "Percent Running Total Weight Read"

-- Data Modify values and percentages
-- WeightCreate is assumed to sum to the total weight

    ,tb."WeightModify"
    ,ROUND(tb."CompletionModify",2) AS "CompletionModify"
    ,ROUND(tb."CompletionModifyWeight",2) AS "CompletionModifyWeight"

    ,
    (
    SELECT
    ROUND(
    SUM("CompletionModifyWeight")
    ,2)
    FROM "tbSCurveStep2"
    WHERE "tbSCurveStep2"."PIDDataset" <= tb."PIDDataset")
    AS "Running Total Weight Modify"

    , ROUND(
    100*(SELECT SUM("CompletionModifyWeight")
    /
    (SELECT(SUM("WeightCreate")) FROM "tbSCurveStep2")
    FROM "tbSCurveStep2"
    WHERE "tbSCurveStep2"."PIDDataset" <= tb."PIDDataset"
    AND
    "tbSCurveStep2"."LogicalStep" = tb."LogicalStep"
    )
    ,2)
    AS "Percent Running Total Weight Modify"

-- Total Create/Read/Modify Incremental Percentages

    , ROUND(
    100*(SELECT(
    SUM("CompletionCreateWeight")
    + SUM("CompletionReadWeight")
    + SUM("CompletionModifyWeight")
    )
    /
    (SELECT(SUM("WeightCreate")) FROM "tbSCurveStep2")
    FROM "tbSCurveStep2"
    WHERE "tbSCurveStep2"."PIDDataset" <= tb."PIDDataset"
    AND
    "tbSCurveStep2"."LogicalStep" = tb."LogicalStep"
    )
    ,2)
    AS "Percent Running Total Weight"

-- Total Create/Read/Modify Total Percentages per Step

    , ROUND(
    100*(SELECT(
    SUM("CompletionCreateWeight")
    + SUM("CompletionReadWeight")
    + SUM("CompletionModifyWeight")
    )
    /
    (SELECT(SUM("WeightCreate")) FROM "tbSCurveStep2")
    FROM "tbSCurveStep2"
    WHERE
    "tbSCurveStep2"."LogicalStep" <= tb."LogicalStep"
    )
    ,2)
    AS "Percent Total Weight"

INTO TABLE "tbSCurvePercentage"

FROM "tbSCurveStep2" AS tb

-- ORDER BY tb."DatasetName" ASC
ORDER BY tb."LogicalStep" ASC
;

SELECT * FROM "tbSCurvePercentage";

-- PSQL COPY only for superusers use \copy

```

```
-- COPY "tbSCurvePercentage" TO 'tbSCurvePercentage.csv' DELIMITER ',';
\!rm tbSCurvePercentage.csv
\copy "tbSCurvePercentage" TO 'tbSCurvePercentage.csv' DELIMITER ','
\end{boxedverbatim}
\normalsize

\subsection*{File on Linux system output in .csv format}
\scriptsize
\begin{boxedverbatim}
Step0001 ,5.00,3.47,0.00,0.00,3.47,3.47
Step0002 ,8.00,5.56,5.00,3.47,9.03,12.50
Step0003 ,13.40,9.31,40.50,28.13,37.43,49.93
Step0004 ,2.40,1.67,56.75,39.41,41.08,91.01
Step0005 ,0.00,0.00,12.35,8.58,8.58,99.58
Step0006 ,0.00,0.00,0.60,0.42,0.42,100.00
```

PSQL Screen output on Linux SSH Terminal

```
javbs> \i SCurveStepPctSummary.sql

                                version
-----
PostgreSQL 8.1.4 on i386-redhat-linux-gnu, compiled by GCC i386-redhat-linux-gcc (GCC) 4.1.0 20060304 (Red Hat 4.1.0-3)
(1 row)

      date
      ----
2006-11-03
(1 row)

DROP TABLE
SELECT
LogicalStep | RunTotWghtCreate | %RunTotWghtCreate | RunTotWghtModify | %RunTotWghtModify | %RunTotWght | %TotWght
-----+-----+-----+-----+-----+-----+-----
Step0001 | 5.00 | 3.47 | 0.00 | 0.00 | 3.47 | 3.47
Step0002 | 8.00 | 5.56 | 5.00 | 3.47 | 9.03 | 12.50
Step0003 | 13.40 | 9.31 | 40.50 | 28.13 | 37.43 | 49.93
Step0004 | 2.40 | 1.67 | 56.75 | 39.41 | 41.08 | 91.01
Step0005 | 0.00 | 0.00 | 12.35 | 8.58 | 8.58 | 99.58
Step0006 | 0.00 | 0.00 | 0.60 | 0.42 | 0.42 | 100.00
(6 rows)
```

J.12 Using Microsoft Data Access Pages

Data Access Pages can only be created using Access 2000 (or later) and can only be viewed by users of Microsoft Internet Explorer 5.0 (or later).

Any Access 2000 and later table, query and report can be saved as a data access page which is accessible form via the HTTP protocol.

The .htm file generated contains VBScript code.

Refer to:

```
http://www.microsoft.com/downloads/details.aspx?amp;
displaylang=en&familyid=982B0359-0A86-4FB2-A7EE-5F3A499515DD&displaylang=en&HelpLCID=1033
```

There a number of security issues with domains of users and file permissions which need to be set up carefully for remote users to pick up the data access pages which are in the form of scripts contained in .htm files.

The connection string in these files must contain the full name of the host reference for web browser (Microsoft Internet Explorer Version 5 or later) on a remote Windows PC to display the data referred to in the access page.

Example: \\sivjavbs edited in place of C:\ provided by default data access page.

It seems as if the display in the browser does not work if the file is opened with the browser directly. When one 'double clicks' on the file in the Windows Explorer display it opens in the browser and the display is shown.

```
<a:ConnectionString>Provider=Microsoft.Jet.OLEDB.4.0;
User ID=Admin;Data Source=\\sivjavbs\My Documents\Projekte\EngineeringProcessModel\ODBCpSQLEngProcess.mdb;
Mode=Share Deny None;
```

```

Extended Properties=&quot;&quot;&quot;;
Persist Security Info=False;
Jet OLEDB:System database=&quot;&quot;&quot;&quot;;
Jet OLEDB:Registry Path=&quot;&quot;&quot;&quot;;
Jet OLEDB:Database Password=&quot;&quot;&quot;&quot;&quot;;
Jet OLEDB:Engine Type=0;
Jet OLEDB:Database Locking Mode=1;
Jet OLEDB:Global Partial Bulk Ops=2;
Jet OLEDB:Global Bulk Transactions=1;
Jet OLEDB:New Database Password=&quot;&quot;&quot;&quot;&quot;;
Jet OLEDB:Create System Database=False;
Jet OLEDB:Encrypt Database=False;
Jet OLEDB:Don't Copy Locale on Compact=False;
Jet OLEDB:Compact Without Replica Repair=False;
Jet OLEDB:SFP=False</a:ConnectionString>&#13;&#10;

```

A typical data access page .htm file content is listed below:
 Contents of tbSCurveStepPctSummary.htm

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML xmlns="http://www.w3.org/TR/REC-html40" xmlns:o =
"urn:schemas-microsoft-com:office:office" xmlns:a =
"urn:schemas-microsoft-com:office:access" xmlns:x =
"urn:schemas-microsoft-com:office:excel" xmlns:dt =
"uuid:C2F41010-65B3-11d1-A29F-00AA00C14882"><HEAD><TITLE>tbSCurveStepPctSummary</TITLE>
<META content="HTML 4.0" name=vs_targetSchema><LINK
href="tbSCurveStepPctSummary_files/filelist.xml" type=text/xml rel=File-List>
<META content=Access.Application name=ProgId>
<META name=VBSForEventHandlers value="true">
<META http-equiv=Content-Type content=text/html; charset=UTF-8>
<META content=10.00.2225 name=DesignerVersion>
<OBJECT id=MSODSC tabIndex=-1
classid=CLSID:0002E553-0000-0000-C000-000000000046>
<PARAM NAME="XMLData" VALUE="<xml xmlns:a=&quot;urn:schemas-microsoft-com:office:access&quot;;
&#13;&#10;<a:DataSourceControl>&#13;&#10;
<a:OWCVersion>10.0.0.6619
</a:OWCVersion>&#13;&#10;
<a:ConnectionString>Provider=Microsoft.Jet.OLEDB.4.0;
User ID=Admin;
Data Source=\\sivjavbs\My Documents\Projekte\EngineeringProcessModel\ODBCpSQLEngProcess.mdb;
Mode=Share Deny None;Extended Properties=&quot;
&quot;&quot;&quot;&quot;;
Persist Security Info=False;
Jet OLEDB:System database=&quot;&quot;&quot;&quot;&quot;;
Jet OLEDB:Registry Path=&quot;&quot;&quot;&quot;&quot;;
Jet OLEDB:Database Password=&quot;&quot;&quot;&quot;&quot;&quot;;
Jet OLEDB:Engine Type=0;
Jet OLEDB:Database Locking Mode=1;
Jet OLEDB:Global Partial Bulk Ops=2;
Jet OLEDB:Global Bulk Transactions=1;
Jet OLEDB:New Database Password=&quot;&quot;&quot;&quot;&quot;&quot;;
Jet OLEDB:Create System Database=False;
Jet OLEDB:Encrypt Database=False;
Jet OLEDB:Don't Copy Locale on Compact=False;
Jet OLEDB:Compact Without Replica Repair=False;
Jet OLEDB:SFP=False</a:ConnectionString>&#13;&#10;
<a:MaxRecords>10000</a:MaxRecords>&#13;&#10;
<a:GridX>24</a:GridX>&#13;&#10;
<a:GridY>24</a:GridY>&#13;&#10;
<a:OfflineType>2</a:OfflineType>&#13;&#10;
<a:XMLLocation>0</a:XMLLocation>&#13;&#10;
<a:XMLDataTarget></a:XMLDataTarget>&#13;&#10;
<a:ConnectionFile></a:ConnectionFile>&#13;&#10;
<a:ElementExtension>&#13;&#10;
<a:ElementID>LogicalStep</a:ElementID>&#13;&#10;
<a:ControlSource>LogicalStep</a:ControlSource>&#13;&#10;
<a:ChildLabel>LogicalStep_Label</a:ChildLabel>&#13;&#10;
</a:ElementExtension>&#13;&#10;
<a:ElementExtension>&#13;&#10;
<a:ElementID>RunTotWghtCreate</a:ElementID>&#13;&#10;
<a:ControlSource>RunTotWghtCreate</a:ControlSource>&#13;&#10;
<a:ChildLabel>RunTotWghtCreate_Label</a:ChildLabel>&#13;&#10;
</a:ElementExtension>&#13;&#10;
<a:ElementExtension>&#13;&#10;
<a:ElementID>%RunTotWghtCreate</a:ElementID>&#13;&#10;
<a:ControlSource>%RunTotWghtCreate</a:ControlSource>&#13;&#10;
<a:ChildLabel>%RunTotWghtCreate_Label</a:ChildLabel>&#13;&#10;
</a:ElementExtension>&#13;&#10;
<a:ElementExtension>&#13;&#10;
<a:ElementID>RunTotWghtModify</a:ElementID>&#13;&#10;
<a:ControlSource>RunTotWghtModify</a:ControlSource>&#13;&#10;

```



```

<a:ChildLabel>RunTotWghtModify_Label</a:ChildLabel>&#13;&#10;
</a:ElementExtension>&#13;&#10; <a:ElementExtension>&#13;&#10;
<a:ElementID>%RunTotWghtModify</a:ElementID>&#13;&#10;
<a:ControlSource>%RunTotWghtModify</a:ControlSource>&#13;&#10;
<a:ChildLabel>%RunTotWghtModify_Label</a:ChildLabel>&#13;&#10;
</a:ElementExtension>&#13;&#10; <a:ElementExtension>&#13;&#10;
<a:ElementID>%RunTotWght</a:ElementID>&#13;&#10;
<a:ControlSource>%RunTotWght</a:ControlSource>&#13;&#10;
<a:ChildLabel>%RunTotWght_Label</a:ChildLabel>&#13;&#10;
</a:ElementExtension>&#13;&#10; <a:ElementExtension>&#13;&#10;
<a:ElementID>%TotWght</a:ElementID>&#13;&#10;
<a:ControlSource>%TotWght</a:ControlSource>&#13;&#10;
<a:ChildLabel>%TotWght_Label</a:ChildLabel>&#13;&#10;
</a:ElementExtension>&#13;&#10;
<a:ElementExtension>&#13;&#10;
<a:ElementID>tbSCurveStepPctSummaryNavLabel</a:ElementID>&#13;&#10;
<a:RecordsetLabel>tbSCurveStepPctSummary |0 of |2;tbSCurveStepPctSummary |0-|1 of |2</a:RecordsetLabel>&#13;&#10;
</a:ElementExtension>&#13;&#10;
<a:GroupLevel>&#13;&#10;
<a:RecordSource>tbSCurveStepPctSummary</a:RecordSource>&#13;&#10;
<a:DefaultSort></a:DefaultSort>&#13;&#10;
<a:HeaderElementId>HeadertbSCurveStepPctSummary</a:HeaderElementId>&#13;&#10;
<a:FooterElementId></a:FooterElementId>&#13;&#10;
<a:CaptionElementId>CaptiontbSCurveStepPctSummary</a:CaptionElementId>&#13;&#10;
<a:RecordNavigationElementId>NavigationtbSCurveStepPctSummary</a:RecordNavigationElementId>&#13;&#10;
<a:DataPageSize>10</a:DataPageSize>&#13;&#10;
<a:GroupFilterControl></a:GroupFilterControl>&#13;&#10;
<a:RecordSelector>/>&#13;&#10; </a:GroupLevel>&#13;&#10;
<a:Datamodel a:version=&quot;0816&quot;>&#13;&#10;
<a:SchemaRowsource a:id=&quot;tbSCurveStepPctSummary&quot; a:type=&quot;dscTable&quot;>&#13;&#10;
<a:SchemaField a:id=&quot;LogicalStep&quot; a:datatype=&quot;130&quot; a:size=&quot;255&quot;>/>&#13;&#10;
<a:SchemaField a:id=&quot;RunTotWghtCreate&quot; a:datatype=&quot;5&quot; a:size=&quot;0&quot;>/>&#13;&#10;
<a:SchemaField a:id=&quot;%RunTotWghtCreate&quot; a:datatype=&quot;5&quot; a:size=&quot;0&quot;>/>&#13;&#10;
<a:SchemaField a:id=&quot;RunTotWghtModify&quot; a:datatype=&quot;5&quot; a:size=&quot;0&quot;>/>&#13;&#10;
<a:SchemaField a:id=&quot;%RunTotWghtModify&quot; a:datatype=&quot;5&quot; a:size=&quot;0&quot;>/>&#13;&#10;
<a:SchemaField a:id=&quot;%RunTotWght&quot; a:datatype=&quot;5&quot; a:size=&quot;0&quot;>/>&#13;&#10;
<a:SchemaField a:id=&quot;%TotWght&quot; a:datatype=&quot;5&quot; a:size=&quot;0&quot;>/>&#13;&#10;
</a:SchemaRowsource>&#13;&#10; <a:RecordsetDef a:id=&quot;tbSCurveStepPctSummary&quot;>&#13;&#10;
<a:PageField a:id=&quot;LogicalStep&quot;>/>&#13;&#10;
<a:PageField a:id=&quot;RunTotWghtCreate&quot;>/>&#13;&#10;
<a:PageField a:id=&quot;%RunTotWghtCreate&quot;>/>&#13;&#10;
<a:PageField a:id=&quot;RunTotWghtModify&quot;>/>&#13;&#10;
<a:PageField a:id=&quot;%RunTotWghtModify&quot;>/>&#13;&#10;
<a:PageField a:id=&quot;%RunTotWght&quot;>/>&#13;&#10;
<a:PageField a:id=&quot;%TotWght&quot;>/>&#13;&#10;
</a:RecordsetDef>&#13;&#10;
</a:Datamodel>&#13;&#10;
</a:DataSourceControl>&#13;&#10;</xml>>
</OBJECT>

```

```

<STYLE id=MSODAPDEFAULTS type=text/css>.MSTheme-Label {
    BORDER-RIGHT: 0px; PADDING-RIGHT: 3px; BORDER-TOP: 0px; PADDING-
    LEFT: 3px; FONT-SIZE: 8pt; OVERFLOW: visible;
    BORDER-LEFT: 0px; WIDTH: 1in; BORDER-BOTTOM: 0px; FONT-FAMILY:
    Tahoma; HEIGHT: 0.156in; TEXT-ALIGN: left
}
.MsoTextbox {
    PADDING-RIGHT: 3px; PADDING-LEFT: 3px; FONT-SIZE: 8pt; OVERFLOW:
    hidden; WIDTH: 1in; FONT-FAMILY: Tahoma; HEIGHT:
    0.197in
}
.MsoBoundSpan {
    BORDER-RIGHT: 0px; PADDING-RIGHT: 3px; BORDER-TOP: 0px; PADDING-
    LEFT: 3px; FONT-SIZE: 8pt; OVERFLOW: hidden;
    BORDER-LEFT: 0px; BORDER-BOTTOM: 0px; FONT-FAMILY: Tahoma; TEXT-
    ALIGN: left
}
.MsoHyperlinkDisplayText {
    BORDER-RIGHT: 0px; PADDING-RIGHT: 3px; BORDER-TOP: 0px; PADDING-
    LEFT: 3px; FONT-SIZE: 8pt; OVERFLOW: hidden;
    BORDER-LEFT: 0px; CURSOR: hand; BORDER-BOTTOM: 0px; FONT-FAMILY:
    Tahoma; TEXT-ALIGN: left
}
.Mso2dSection {
    LEFT: 0px; BEHAVIOR: url(#DEFAULT#Mso2dSection); OVERFLOW: hidden;
    POSITION: relative; TOP: 0px; BACKGROUND-COLOR:
    transparent
}
.Mso2dSectionBanner {
    PADDING-RIGHT: 4px; DISPLAY: none; PADDING-LEFT: 4px; FONT-WEIGHT:
    normal; FONT-SIZE: 8pt; LEFT: 0px; BEHAVIOR:

```

```

        url(#DEFAULT#Mso2dSectionBanner); PADDING-TOP: 2px; FONT-FAMILY:
        Tahoma; TOP: 0px; HEIGHT: 0.2in; BACKGROUND-
        COLOR: buttonface
    }
    .MsoRectangle {
        BORDER-RIGHT: black 1px solid; BORDER-TOP: black 1px solid;
        OVERFLOW: hidden; BORDER-LEFT: black 1px solid;
        BORDER-BOTTOM: black 1px solid
    }
    .MsoTitle {
        DISPLAY: none; FONT-WEIGHT: normal; COLOR: inactivecaptiontext
    }
    .MsoExpandCollapse {
        CURSOR: hand
    }
    .MsoNavContainer {
        BORDER-RIGHT: gainsboro 1px solid; BORDER-TOP: gainsboro 1px solid;
        Z-INDEX: -1; BORDER-LEFT: gainsboro 1px solid;
        CURSOR: hand; BORDER-BOTTOM: gainsboro 1px solid; POSITION:
        absolute; HEIGHT: 25px; BACKGROUND-COLOR: gainsboro
    }
    .MsoNavButton {
        BORDER-RIGHT: gainsboro 1px solid; BORDER-TOP: gainsboro 1px solid;
        BORDER-LEFT: gainsboro 1px solid; CURSOR:
        auto; BORDER-BOTTOM: gainsboro 1px solid; BACKGROUND-REPEAT: no-
        repeat
    }
    .MsoNavButtonMouseOver {
        BORDER-RIGHT: highlight 1px solid; BORDER-TOP: highlight 1px solid;
        BORDER-LEFT: highlight 1px solid; BORDER-
        BOTTOM: highlight 1px solid; BACKGROUND-COLOR: buttonhighlight
    }
    .MsoNavToggleButtonMouseOver {
        BORDER-RIGHT: highlight 1px solid; BORDER-TOP: highlight 1px solid;
        BORDER-LEFT: highlight 1px solid; BORDER-
        BOTTOM: highlight 1px solid; BACKGROUND-REPEAT: no-repeat;
        BACKGROUND-COLOR: buttonhighlight
    }
    .MsoNavButtonMouseDown {
        BORDER-RIGHT: buttonshadow 1px solid; BORDER-TOP: buttonshadow 1px
        solid; BORDER-LEFT: buttonshadow 1px solid;
        CURSOR: hand; BORDER-BOTTOM: buttonshadow 1px solid; BACKGROUND-
        COLOR: buttonshadow
    }
    .MsoNavRecordsetLabel {
        BORDER-RIGHT: gainsboro 1px solid; BORDER-TOP: gainsboro 1px solid;
        PADDING-LEFT: 0px; FONT-SIZE: 8pt; OVERFLOW:
        hidden; BORDER-LEFT: gainsboro 1px solid; WIDTH: 100%; CURSOR:
        default; PADDING-TOP: 0px; BORDER-BOTTOM: gainsboro
        1px solid; FONT-FAMILY: Tahoma
    }
    .MsoRecordSelector {
        BORDER-RIGHT: buttonshadow 1px solid; BORDER-TOP: buttonshadow 1px
        solid; OVERFLOW: hidden; BORDER-LEFT:
        buttonshadow 1px solid; WIDTH: 0.17in; BORDER-BOTTOM: buttonshadow
        1px solid; HEIGHT: 100%; BACKGROUND-COLOR:
        gainsboro
    }
    .MsoRecordSelectorCurrent {
        BACKGROUND-POSITION: 1px 0px; BACKGROUND-IMAGE:
        url(owc://GIF/#11240); BACKGROUND-REPEAT: no-repeat
    }
    .MsoRecordSelectorSelectedImage {
        BACKGROUND-POSITION: 1px 0px; BACKGROUND-IMAGE:
        url(owc://GIF/#11241); BACKGROUND-REPEAT: no-repeat
    }
    .MsoRecordSelectorSelected {
        BORDER-RIGHT: buttonshadow 1px solid; BORDER-TOP: buttonshadow 1px
        solid; LEFT: 0px; OVERFLOW: hidden; BORDER-
        LEFT: buttonshadow 1px solid; WIDTH: 0.17in; BORDER-BOTTOM:
        buttonshadow 1px solid; TOP: 0px; HEIGHT: 100%;
        BACKGROUND-COLOR: buttonshadow; POSTION: relative
    }
    .MsoRecordSelectorDirty {
        BACKGROUND-POSITION: -1px 0px; BACKGROUND-IMAGE:
        url(owc://GIF/#11242); BACKGROUND-REPEAT: no-repeat
    }
    .MsoRecordSelectorTransparent {
        LEFT: 0px; OVERFLOW: hidden; WIDTH: 0.17in; TOP: 0px; POSTION:
        relative
    }
    HR {
        COLOR: black
    }
    SELECT {

```

```

        FONT-SIZE: 8pt; FONT-FAMILY: Tahoma
    }
    INPUT {
        FONT-SIZE: 8pt; FONT-FAMILY: Tahoma
    }
    BODY {
        FONT-SIZE: 10pt; FONT-FAMILY: Tahoma
    }
    MARQUEE {
        FONT-SIZE: 8pt; FONT-FAMILY: Tahoma
    }
    LEGEND {
        FONT-SIZE: 8pt; FONT-FAMILY: Tahoma
    }
    BUTTON {
        FONT-SIZE: 8pt; FONT-FAMILY: Tahoma
    }
    TEXTAREA {
        FONT-SIZE: 8pt; FONT-FAMILY: Tahoma
    }
    .MSODatasheetText {
        FONT-WEIGHT: 300; FONT-SIZE: 10pt; COLOR: #000000; FONT-STYLE:
        normal; FONT-FAMILY: Arial; TEXT-DECORATION: none
    }
}
</STYLE>

```

```

<META content="MSHTML 6.00.2900.2873" name=GENERATOR>
<SCRIPT language=Javascript id=MSODSC_Validation>
validateBrowser();

function validateBrowser() {
    strVers=navigator.appVersion
    strName=navigator.appName
    strPlat=navigator.platform
    intIndex1=strVers.indexOf("MSIE");
    intIndex1=intIndex1+5
    intIndex2=strVers.lastIndexOf(";");
    intVer=strVers.substring(intIndex1, intIndex2)
    intVer=parseInt(intVer)
    if (strName=="Microsoft Internet Explorer" && strPlat=="Win32" &&
        intVer>="5") {
        validateOWC();
    }
    else {
        strMsgGetIE="<TABLE cellSpacing=0 cellPadding=0 width='95%'
        border=0 height='8'><TR>"
        strMsgGetIE+="<TD bgColor='#336699' height=25 width=15>
        &nbsp;</TD><TD bgColor='#666666' width=500px><FONT
        face=Tahoma "
        strMsgGetIE+="size=4 color=white><b>&nbsp;&nbsp;&nbsp;&nbsp;&Data Access Page
        Notification</B></FONT></TD></TR>"
        strMsgGetIE+="<TR><TD bgColor='#cccccc' width=15>
        &nbsp;</TD><TD bgColor='#cccccc' width=500px><BR>"
        strMsgGetIE+="<p><font face='Tahoma' size='2'>"
        strMsgGetIE+="This page requires Windows IE 5.0 or
        higher.</p>"
        strMsgGetIE+="<a href='

        http://www.microsoft.com/isapi/redir.dll?Prd=Office&Sbp=
        Access&Pver=10&Ar=DPdesigner&Sba=IEhome&Plcid=1033
        '><p align='center'>"
        strMsgGetIE+="Click here to install the latest version of
        Internet Explorer.</a></font></p><br></TD></TR>
        </TABLE>"
        document.write(strMsgGetIE)
    }
}

function validateOWC() {
    if (MSODSC.object==null) {
        strMsgGetOWC="<TABLE width='95%' cellpadding=0 cellspacing=
        0 border=0 height='8'>"
        strMsgGetOWC+="<TR><TD bgColor='#336699' height=25 width=
        15>&nbsp;</TD><TD bgColor='#666666' width=500px>"
        strMsgGetOWC+="<FONT face=Tahoma color=white size=4><B>
        &nbsp;&nbsp;&nbsp;&nbsp;&&nbsp;&nbsp;&nbsp;&nbsp;&Data Access Page Notification</B></FONT>
        </TD></TR><TR><TD bgColor='#cccccc' width=15>
        &nbsp;</TD><TD bgColor='#cccccc' width=500px><BR>"
        strMsgGetOWC+="<p><font face='Tahoma' size='2'>This page
        requires the Microsoft Office Web
    }
}

```

```

Components.</p>"
strMsgGetOWC+="</font><p><font face='Tahoma' size='2'>See
the <a href='

http://office.microsoft.com/office/redirect/10/MSOWCPub.asp
?&HelpLCID=1033'>Microsoft Office Web
site</a> for more information. "
strMsgGetOWC+="</font></p></TD></TR></TABLE>"
document.write(strMsgGetOWC)
}
}

```

```

</SCRIPT>
<!--[if gte mso 9]><xml>
<o:DocumentProperties>
  <o:LastAuthor>JA vanB Strasheim</o:LastAuthor>
  <o:Revision>3</o:Revision>
  <o:TotalTime>16</o:TotalTime>
  <o:LastSaved>2006-06-08T15:52:38Z</o:LastSaved>
  <o:Version>10.6735</o:Version>
</o:DocumentProperties>
<o:OfficeDocumentSettings>
  <o:DownloadComponents/>
  <o:LocationOfComponents href="file:///\\\"/>
</o:OfficeDocumentSettings>
</xml><![endif]-->
</HEAD>
<BODY style="MARGIN: 0px; OVERFLOW: auto" vLink=#800080 link=#0000ff>
<DIV class=Mso2dSectionBanner id=CaptiontbSCurveStepPctSummaryBanner
style="WIDTH: 20.946cm; tabIndex=-1"><SPAN
id=CaptiontbSCurveStepPctSummaryBannerCaption>Caption:
tbSCurveStepPctSummary</SPAN></DIV>
<DIV class=Mso2dSection id=CaptiontbSCurveStepPctSummary
style="WIDTH: 20.946cm; BORDER-BOTTOM: #c0c0c0 1px solid; HEIGHT: 17px">
<SPAN
class="MSTheme-Label MSODatasheetText" id=LogicalStep_Label
style="LEFT: 0cm; OVERFLOW: hidden; WIDTH: 2.38cm; POSITION: absolute;
HEIGHT: 1.3em; TEXT-ALIGN: center"
MsoTextAlign="General">LogicalStep </SPAN>
<SPAN
class="MSTheme-Label MSODatasheetText" id=RunTotWghtCreate_Label
style="LEFT: 2.38cm; OVERFLOW: hidden; WIDTH: 3.227cm; POSITION: absolute;
HEIGHT: 1.3em; TEXT-ALIGN: center"
MsoTextAlign="General">RunTotWghtCreate </SPAN>
<SPAN
class="MSTheme-Label MSODatasheetText" id=%RunTotWghtCreate_Label
style="LEFT: 5.606cm; OVERFLOW: hidden; WIDTH: 3.386cm; POSITION: absolute;
HEIGHT: 1.3em; TEXT-ALIGN: center"
MsoTextAlign="General">%RunTotWghtCreate </SPAN>
<SPAN
class="MSTheme-Label MSODatasheetText" id=RunTotWghtModify_Label
style="LEFT: 8.992cm; OVERFLOW: hidden; WIDTH: 3.65cm; POSITION: absolute;
HEIGHT: 1.3em; TEXT-ALIGN: center"
MsoTextAlign="General">RunTotWghtModify </SPAN>
<SPAN
class="MSTheme-Label MSODatasheetText" id=%RunTotWghtModify_Label
style="LEFT: 12.642cm; OVERFLOW: hidden; WIDTH: 3.544cm; POSITION:
absolute; HEIGHT: 1.3em; TEXT-ALIGN: center"
MsoTextAlign="General">%RunTotWghtModify </SPAN>
<SPAN
class="MSTheme-Label MSODatasheetText" id=%RunTotWght_Label
style="LEFT: 16.186cm; OVERFLOW: hidden; WIDTH: 2.38cm; POSITION: absolute;
HEIGHT: 1.3em; TEXT-ALIGN: center"
MsoTextAlign="General">%RunTotWght </SPAN>
<SPAN
class="MSTheme-Label MSODatasheetText" id=%TotWght_Label
style="LEFT: 18.566cm; OVERFLOW: hidden; WIDTH: 2.38cm; POSITION: absolute;
HEIGHT: 1.3em; TEXT-ALIGN: center"
MsoTextAlign="General">%TotWght </SPAN>
</DIV>
<DIV class=Mso2dSectionBanner id=HeadertbSCurveStepPctSummaryBanner
style="WIDTH: 20.946cm; tabIndex=-1">
<SPAN
id=HeadertbSCurveStepPctSummaryBannerCaption>Header:
tbSCurveStepPctSummary</SPAN>
</DIV>
<DIV class=Mso2dSection id=HeadertbSCurveStepPctSummary
style="WIDTH: 20.946cm; HEIGHT: 17px; BACKGROUND-COLOR: #ffffff">
<TEXTAREA class="MsoTextbox MSODatasheetText" id=
LogicalStep style="BORDER-RIGHT: #c0c0c0 1px solid; BORDER-TOP: 0px; LEFT:
0cm; OVERFLOW: hidden; BORDER-LEFT: 0px; WIDTH:
2.38cm; COLOR: #000000; BORDER-BOTTOM: #c0c0c0 1px solid; POSITION:

```

```

absolute; HEIGHT: 1.3em; BACKGROUND-COLOR: #ffffff"
MsoTextAlign="General"></TEXTAREA>
<TEXTAREA class="MsoTextbox MSODatasheetText" id=RunTotWghtCreate
style="BORDER-RIGHT: #c0c0c0 1px solid; BORDER-TOP: 0px;
LEFT: 2.38cm; OVERFLOW: hidden; BORDER-LEFT: 0px; WIDTH: 3.227cm; COLOR:
#000000; BORDER-BOTTOM: #c0c0c0 1px solid;
POSITION: absolute; HEIGHT: 1.3em; BACKGROUND-COLOR: #ffffff"
MsoTextAlign="General">
</TEXTAREA>
<TEXTAREA class="MsoTextbox MSODatasheetText" id=%RunTotWghtCreate
style="BORDER-RIGHT: #c0c0c0 1px solid; BORDER-TOP:
0px; LEFT: 5.606cm; OVERFLOW: hidden; BORDER-LEFT: 0px; WIDTH: 3.386cm;
COLOR: #000000; BORDER-BOTTOM: #c0c0c0 1px solid;
POSITION: absolute; HEIGHT: 1.3em; BACKGROUND-COLOR: #ffffff"
MsoTextAlign="General">
</TEXTAREA>
<TEXTAREA class="MsoTextbox MSODatasheetText" id=RunTotWghtModify
style="BORDER-RIGHT: #c0c0c0 1px solid; BORDER-TOP: 0px;
LEFT: 8.992cm; OVERFLOW: hidden; BORDER-LEFT: 0px; WIDTH: 3.65cm; COLOR:
#000000; BORDER-BOTTOM: #c0c0c0 1px solid;
POSITION: absolute; HEIGHT: 1.3em; BACKGROUND-COLOR: #ffffff"
MsoTextAlign="General">
</TEXTAREA>
<TEXTAREA class="MsoTextbox MSODatasheetText" id=%RunTotWghtModify
style="BORDER-RIGHT: #c0c0c0 1px solid; BORDER-TOP:
0px; LEFT: 12.642cm; OVERFLOW: hidden; BORDER-LEFT: 0px; WIDTH: 3.544cm;
COLOR: #000000; BORDER-BOTTOM: #c0c0c0 1px solid;
POSITION: absolute; HEIGHT: 1.3em; BACKGROUND-COLOR: #ffffff"
MsoTextAlign="General">
</TEXTAREA>
<TEXTAREA class="MsoTextbox MSODatasheetText" id=%RunTotWght style="BORDER-
RIGHT: #c0c0c0 1px solid; BORDER-TOP: 0px;
LEFT: 16.186cm; OVERFLOW: hidden; BORDER-LEFT: 0px; WIDTH: 2.38cm; COLOR:
#000000; BORDER-BOTTOM: #c0c0c0 1px solid;
POSITION: absolute; HEIGHT: 1.3em; BACKGROUND-COLOR: #ffffff"
MsoTextAlign="General">
</TEXTAREA>
<TEXTAREA class="MsoTextbox MSODatasheetText" id=%TotWght style="BORDER-
RIGHT: #c0c0c0 1px solid; BORDER-TOP: 0px; LEFT:
18.566cm; OVERFLOW: hidden; BORDER-LEFT: 0px; WIDTH: 2.38cm; COLOR:
#000000; BORDER-BOTTOM: #c0c0c0 1px solid; POSITION:
absolute; HEIGHT: 1.3em; BACKGROUND-COLOR: #ffffff" MsoTextAlign="General">
</TEXTAREA>
</DIV>
<DIV class=Mso2dSectionBanner id=NavigationtbSCurveStepPctSummaryBanner
style="WIDTH: 20.946cm" tabIndex=-1><SPAN
id=NavigationtbSCurveStepPctSummaryBannerCaption>Navigation:
tbSCurveStepPctSummary</SPAN></DIV>
<DIV class=Mso2dSection id=NavigationtbSCurveStepPctSummary
style="VISIBILITY: hidden; WIDTH: 20.946cm; HEIGHT: 0.427in">
<TABLE class=MsoNavContainer id=tbSCurveStepPctSummaryNavigation
style="LEFT: 4px; WIDTH: 6in; POSITION: absolute; TOP: 4px" cellSpacing=0
cellPadding=0>
<TBODY>
<TR>
<TD class=MsoNavButton style="WIDTH: 20px; HEIGHT: 20px"><IMG
class=MsoNavFirst id=tbSCurveStepPctSummaryNavFirst tabIndex=1
height=20
src="owc://GIF/#11200" width=20></TD>
<TD class=MsoNavButton style="WIDTH: 20px; HEIGHT: 20px"><IMG
class=MsoNavPrevious id=tbSCurveStepPctSummaryNavPrevious tabIndex=2
height=20 src="owc://GIF/#11202" width=20></TD>
<TD style="VERTICAL-ALIGN: middle; WIDTH: 100%; TEXT-ALIGN: center"
noWrap><SPAN class=MsoNavRecordsetLabel id=
tbSCurveStepPctSummaryNavLabel
style="VISIBILITY: hidden">tbSCurveStepPctSummary |0-|1 of |2</SPAN>
</TD>
<TD class=MsoNavButton style="WIDTH: 20px; HEIGHT: 20px"><IMG
class=MsoNavNext id=tbSCurveStepPctSummaryNavNext tabIndex=4 height=
20
src="owc://GIF/#11204" width=20></TD>
<TD class=MsoNavButton style="WIDTH: 20px; HEIGHT: 20px"><IMG
class=MsoNavLast id=tbSCurveStepPctSummaryNavLast tabIndex=5 height=
20
src="owc://GIF/#11206" width=20></TD>
<TD class=MsoNavButton style="WIDTH: 20px; HEIGHT: 20px"><IMG
class=MsoNavAddNew id=tbSCurveStepPctSummaryNavNew tabIndex=6 height=
20
src="owc://GIF/#11208" width=20></TD>
<TD class=MsoNavButton style="WIDTH: 20px; HEIGHT: 20px"><IMG
class=MsoNavDelete id=tbSCurveStepPctSummaryNavDelete tabIndex=7
height=20
src="owc://GIF/#11210" width=20></TD>
<TD class=MsoNavButton style="WIDTH: 20px; HEIGHT: 20px"><IMG
class=MsoNavSave id=tbSCurveStepPctSummaryNavSave tabIndex=8 height=

```

```

20
src="owc://GIF/#11214" width=20></TD>
<TD class=MsoNavButton style="WIDTH: 20px; HEIGHT: 20px"><IMG
class=MsoNavUndo id=tbSCurveStepPctSummaryNavUndo tabIndex=9 height=
20
src="owc://GIF/#11212" width=20></TD>
<TD class=MsoNavButton style="WIDTH: 20px; HEIGHT: 20px"><IMG
class=MsoNavSortAsc id=tbSCurveStepPctSummaryNavSortAscending
tabIndex=10
height=20 src="owc://GIF/#11216" width=20></TD>
<TD class=MsoNavButton style="WIDTH: 20px; HEIGHT: 20px"><IMG
class=MsoNavSortDesc id=tbSCurveStepPctSummaryNavSortDescending
tabIndex=11 height=20 src="owc://GIF/#11218" width=20></TD>
<TD class=MsoNavButton style="WIDTH: 20px; HEIGHT: 20px"><IMG
class=MsoNavFilter id=tbSCurveStepPctSummaryNavFilterBySelection
tabIndex=12 height=20 src="owc://GIF/#11220" width=20></TD>
<TD class=MsoNavButton style="WIDTH: 20px; HEIGHT: 20px"><IMG
class=MsoNavToggleFilter id=tbSCurveStepPctSummaryNavFilterToggle
tabIndex=13 height=20 src="owc://GIF/#11222" width=20></TD>
<TD class=MsoNavButton style="WIDTH: 20px; HEIGHT: 20px"><IMG
class=MsoNavHelp id=tbSCurveStepPctSummaryNavHelp tabIndex=14 height=
20
src="owc://GIF/#11224" width=20></TD></TR>
</TBODY>
</TABLE>
</DIV>
</BODY>
</HTML>

```

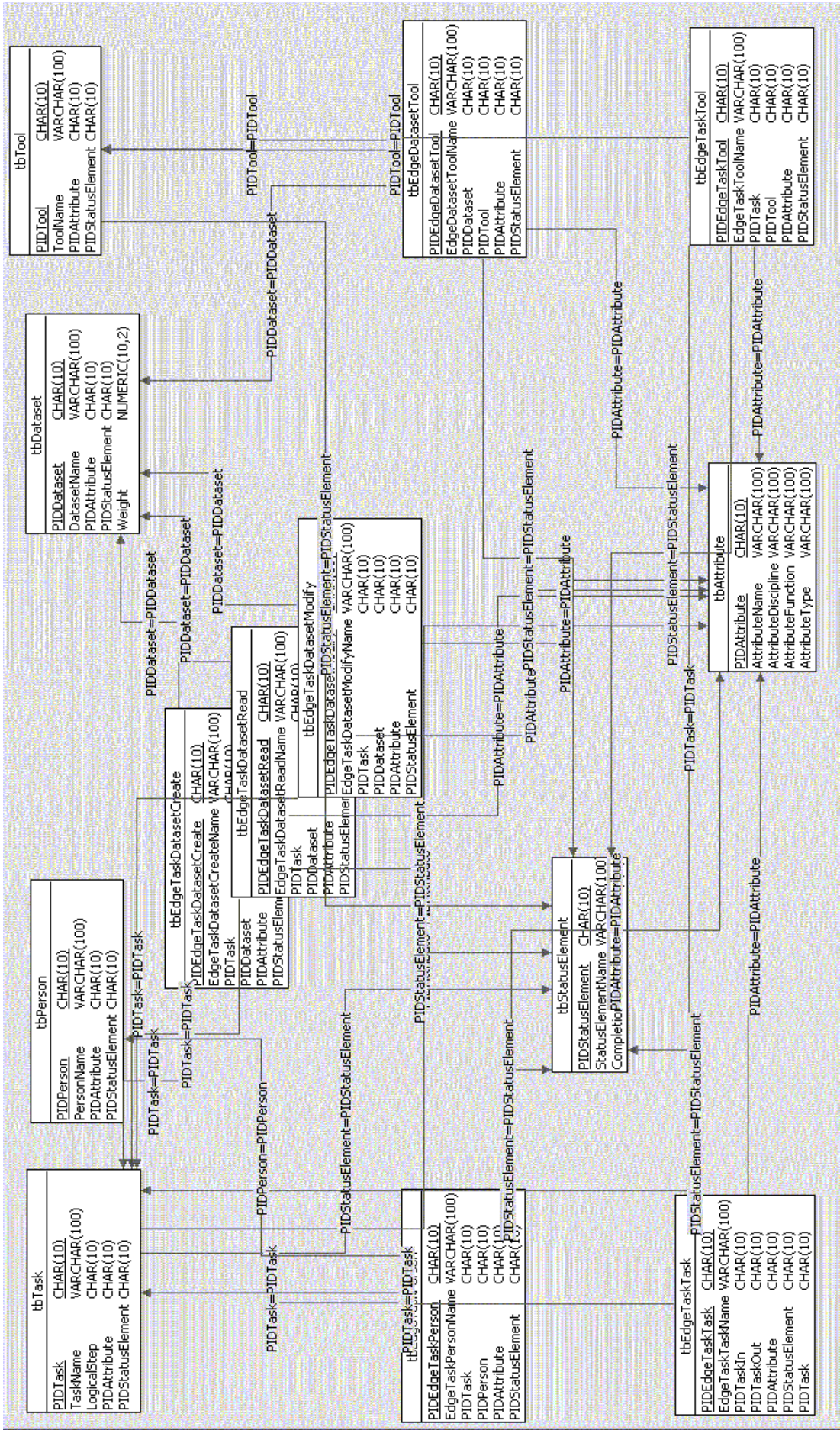



Figure J.1: EngineeringProcessModelClay

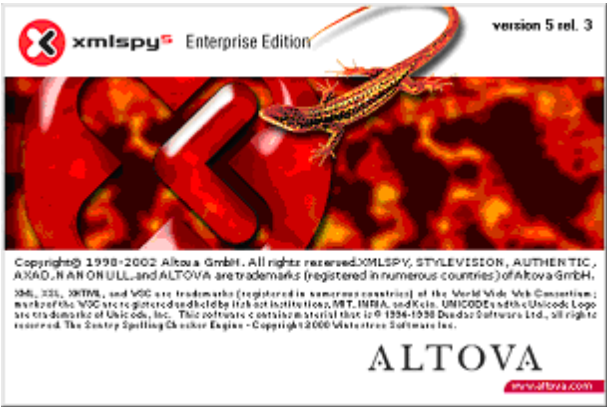


Figure J.2: Altova XMLSPy splash display

Appendix K

Organisation management and reporting structures using graph applications

K.1 Typical Management Reporting Tree Structure

Contents

- Apply graph tree functionality to management structures
- Set up relational algebra functionality
- Adjacency matrix of management structure graph with vertex labels
- Report number of active managers - size of adjacency matrix
- Number of relations between managers - graph edge count
- Test if this is a directed graph
- Number of persons reporting and reported to per manager
- Lengths of paths in management structure adjacency matrix

Apply graph tree functionality to management structures

```
%.....  
% TreesManagementStructures.m  
% Apply graph tree functionality to management structures  
%.....  
clc  
clear all  
format compact
```

Set up relational algebra functionality

```
%.....  
RelationalAlgebraBoolean  
%.....  
% output file prefix  
tgfFilePre='YHMM'
```

```
MATLAB implementation of relational algebra boolean matrix operations in inline functions  
tgfFilePre =  
YHMM
```

Adjacency matrix of management structure graph with vertex labels

Example - seven managers in hierarchy

```

HMM= [ 0 0 0 0 0 0 0 ; ...
       1 0 0 0 0 0 0 ; ...
       1 0 0 0 0 0 0 ; ...
       0 1 0 0 0 0 0 ; ...
       0 1 0 0 0 0 0 ; ...
       0 0 1 0 0 0 0 ; ...
       0 0 1 0 0 0 0 ]
vertexLabels= { ...
    'M_0','M_A','M_B','M_1','M_2','M_3','M_4' ...
}
% Output to yEd data file
isHomog=1
fileRS=horzcat(tgfFilePre,'ReportingStructure','.tgf')
TgfWrite(fileRS,HMM,isHomog,vertexLabels,{});

```

```

HMM =
    0     0     0     0     0     0     0
    1     0     0     0     0     0     0
    1     0     0     0     0     0     0
    0     1     0     0     0     0     0
    0     1     0     0     0     0     0
    0     0     1     0     0     0     0
    0     0     1     0     0     0     0

vertexLabels =
    Columns 1 through 6
    'M_0'    'M_A'    'M_B'    'M_1'    'M_2'    'M_3'
    Column 7
    'M_4'
isHomog =
    1
fileRS =
    YHMMReportingStructure.tgf
File YHMMReportingStructure.tgf opened
nRows =
    7
nCols =
    7
File YHMMReportingStructure.tgf closed

```

Report number of active managers - size of adjacency matrix

Graph vertex count

```
[nRows,nCols]=size(HMM)
```

```

nRows =
    7
nCols =
    7

```

Number of relations between managers - graph edge count

Count number of edges - both directions - normal matrix operations

```

R1= ones(1,size(HMM,2))
C1= ones(size(HMM,2),1)
R_HMM= R1*HMM
R_HMM_C=R_HMM*C1

```

```

R1 =
    1     1     1     1     1     1     1
C1 =
    1
    1
    1
    1
    1
    1
    1
    1
R_HMM =
    2     2     2     0     0     0     0
R_HMM_C =
    6

```

Test if this is a directed graph

not directed - Symmetry - element total = 0

```

T_symm= HMM-HMM'
Test=abs(R1*abs(T_symm*C1))
% Alternative
Test=sum(sum(T_symm))

```

```

T_symm =
    0    -1    -1     0     0     0     0
    1     0     0    -1    -1     0     0
    1     0     0     0     0    -1    -1
    0     1     0     0     0     0     0
    0     1     0     0     0     0     0
    0     0     1     0     0     0     0
    0     0     1     0     0     0     0
Test =
    8
Test =
    0

```

Number of persons reporting and reported to per manager

Number of persons reporting a manager report vertex inDegrees - row sums

```

InD_HMM= R1*HMM
% Number of persons reported to
% report vertex outdegrees - column sums
OutD_HMM=(HMM*C1)'

```

```

InD_HMM =
    2     2     2     0     0     0     0
OutD_HMM =
    0     1     1     1     1     1     1

```

Lengths of paths in management structure adjacency matrix

Paths in A2_3 - vertex degrees also on diagonal

```
A2=HMM*HMM
A3=HMM*A2
A4=HMM*A3
A2b=productR(HMM,HMM)
A3b=productR(A2b,HMM)
```

```
A2 =
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0
    1    0    0    0    0    0    0
    1    0    0    0    0    0    0
    1    0    0    0    0    0    0
    1    0    0    0    0    0    0

A3 =
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0

A4 =
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0

A2b =
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0
    1    0    0    0    0    0    0
    1    0    0    0    0    0    0
    1    0    0    0    0    0    0
    1    0    0    0    0    0    0

A3b =
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0
```

K.2 Converting from adjacency matrix format to adjacency list format

To represent a relation for computational purposes the boolean adjacency-matrix representation, which is more appropriate for dense graphs, or the incidence list representation, which is more appropriate for sparse graphs, can be used.

The MATLAB code for the conversion from adjacency matrix format to adjacency list format is listed below.

```
function [AdjList,nrowL,ncolL] = AdjacencyList(AdjMatrix)
% .....
% Generate adjacency list given adjacency matrix
```

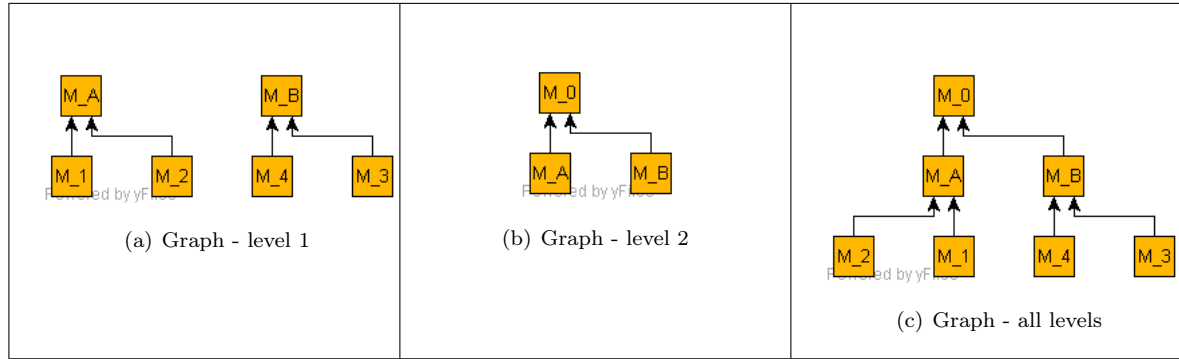


Figure K.1: Reporting or management structure - graph representation - from adjacency matrix: (a) Reporting structure graph representation - level 1; (b) Reporting structure graph representation - level 2; (c) Reporting structure graph representation - all levels;

```
% Adjacency list in matrix format - ignore 0 entries
% .....
AdjList = [];
nrowA = size(AdjMatrix, 1);
ncolA = size(AdjMatrix, 2);
for ir = 1:nrowA
    irowL = ir;
    AdjList(irowL, 1) = ir;
    icolL = 1;
    for ic = 1:ncolA
        if (AdjMatrix(ir, ic) == 1)
            icolL = icolL + 1;
            AdjList(irowL, icolL) = ic;
        end
    end
end
nrowL = size(AdjList, 1);
ncolL = size(AdjList, 2);
```

The MATLAB code for converting from adjacency list format to adjacency matrix format is listed below.

```
function [adjMatrix] = adjMatrixFromList(adjList)
% .....
% Convert graph data representation form adjacency List
% to adjacency Matrix format
% .....
% maximum entry in adjacency list is largest vertex number
nVertices = max(size(adjList, 1), max(max(adjList)));
% set up blank adjacency matrix
adjMatrix = zeros(nVertices);
% adjMatrix = [];
% Loop over vertices and make entries in adjacency matrix
for nv = 1:size(adjList, 1)
    nEdgesRow = size(adjList(nv, :), 2);
    for ne = 2:nEdgesRow
        % end vertex of edge
        mv = adjList(nv, ne);
        % skip zero entries
        if not(mv == 0)
            adjMatrix(nv, mv) = 1;
        end
    end
end
end
```

K.3 Depth first search and tree structure for a given graph

The depth first search algorithm (DFS) can be used to determine the spanning tree for a given graph given in adjacency matrix format. The spanning tree is also output in adjacency matrix format.

To determine the DFS tree the transpose of the adjacency matrix is input. DFS does not produce results on a matrix which is not transposed.

The depth first search algorithm used is contained in the MATLAB code shown below.

Global variables are required due to the recursive nature of the algorithm. A double while loop is required to ensure that unconnected graphs can be processed as well. All vertices need to be investigated before the search terminates.

The *search* function finds the next vertex linked to a given vertex and returns 0 if no vertex can be found.

```
function [tree,dfsArray] = DepthFirstSearch(R)
    global List mark ipos dfsArray
    %.....
    % DFS of graph
    % Input: R - graph in Adjacency matrix format
    % Output dfsArray - traversal of indices in DFS order
    % tree - Graph spanning tree in adjacency matrix format
    %.....
    RelationalAlgebraBoolean
    nVertex=size(R,1);
    [adjList,nrowL,ncolL] = AdjacencyListM(R)
    dfsArray=[];
    % tree=[];
    tree=zeros(size(R));
    List={};
    ipos=0;
    % celldisp(List)
    for v=1:nVertex
        mark(v)=0;
    end
    while (any(mark==0)==1)
        x=min(find(mark==0));
        % disp(['into search x mark: ',num2str(mark)])
        % [List,mark]=search(x,adjList,nrowL,ncolL,mark,List)
        search(x,adjList,nrowL,ncolL);
        % disp(['outof search x mark: ',num2str(mark)])
        while (size(List,2)>0)
            % display(['Size of list = ',num2str(size(List))])
            L=List{end};
            v=L(1);
            w=L(2);
            List=List(1:end-1);
            if (mark(w)==0);
                tree(v,w)=1;
            % disp(['into search v mark: ',num2str(mark)])
            % [List,mark]=search(v,adjList,nrowL,ncolL,mark,List)
            search(w,adjList,nrowL,ncolL);
            % disp(['outof search v mark: ',num2str(mark)])
            % disp(['outof search size List: ',num2str(size(List,2))])
            end
        end
    end

    function []=search(v,adjList,nrowL,ncolL)
        %.....
        % Updates List to include next edge in graph
        % Mark the vertex as processed
        % Update the processing sequence array dfsArray
        % by adding the vertex processed to it
        %.....
        global List mark ipos dfsArray
        dfsArray=[dfsArray,v];
        mark(v)=1;
        for iv=2:ncolL
```

```

        iw=adjList(v,iv);;
        if(iw>0)
            List=[List,[v,iw]];
% disp ([num2str(v),' ',num2str(iw),' Added to List'])
%         celldisp(List);
        end
    end
end

```

Contents

- Testing of Depth First Search Algorithm
- Adjacency matrices of test graphs R
- DFS tested on transpose of adjacency matrices

Testing of Depth First Search Algorithm

global variables

```

global List mark ipos dfsArray
clc
clear all
format compact
%.....
% TestDepthFirstSearch.m
% Depth First Search Algorithm ex Chartrand & Oelerman
% R square relation matrix
%.....

```

RelationalAlgebraBoolean

MATLAB implementation of relational algebra boolean matrix operations in inline functions

Adjacency matrices of test graphs R

```

HMM2= [0 0 0 ; ...
        1 0 0 ; ...
        1 0 0 ]
HMM1= [0 0 0 0 0 0 ; ...
        0 0 0 0 0 0 ; ...
        1 0 0 0 0 0 ; ...
        1 0 0 0 0 0 ; ...
        0 1 0 0 0 0 ; ...
        0 1 0 0 0 0 ]
R=[ 0 1 0 0 0 0 ; ...
    0 0 0 0 1 0 ; ...
    1 1 0 1 0 0 ; ...
    0 1 0 0 1 1 ; ...
    0 0 1 0 0 1 ; ...
    0 0 0 0 0 0 ]
R1=[0 1 1; ...
    0 0 0; ...
    0 0 0]

```

```

HMM2 =
    0     0     0
    1     0     0
    1     0     0

HMM1 =
    0     0     0     0     0     0
    0     0     0     0     0     0
    1     0     0     0     0     0
    1     0     0     0     0     0
    0     1     0     0     0     0
    0     1     0     0     0     0

R =
    0     1     0     0     0     0
    0     0     0     0     1     0
    1     1     0     1     0     0
    0     1     0     0     1     1
    0     0     1     0     0     1
    0     0     0     0     0     0

R1 =
    0     1     1
    0     0     0
    0     0     0

```

DFS tested on transpose of adjacency matrices

```

[tree1,dfsArray1]=DepthFirstSearch(HMM1')
[tree2,dfsArray2]=DepthFirstSearch(HMM2')
[treeR,dfsArrayR]=DepthFirstSearch(R')
[treeR,dfsArrayR]=DepthFirstSearch(R')

```

MATLAB implementation of relational algebra boolean matrix operations in inline functions

```

adjList =
    1     3     4
    2     5     6
    3     0     0
    4     0     0
    5     0     0
    6     0     0

nrowL =
    6

ncoll =
    3

tree1 =
    0     0     1     1     0     0
    0     0     0     0     1     1
    0     0     0     0     0     0
    0     0     0     0     0     0
    0     0     0     0     0     0
    0     0     0     0     0     0

dfsArray1 =
    1     4     3     2     6     5

```

MATLAB implementation of relational algebra boolean matrix operations in inline functions

```

adjList =
    1     2     3
    2     0     0
    3     0     0

nrowL =
    3

ncoll =
    3

tree2 =
    0     1     1
    0     0     0
    0     0     0

dfsArray2 =
    1     3     2

```



```

MATLAB implementation of relational algebra boolean matrix operations in inline functions
adjList =
    1     3     0     0
    2     1     3     4
    3     5     0     0
    4     3     0     0
    5     2     4     0
    6     4     5     0
nrowL =
    6
ncoll =
    4
treeR =
    0     0     1     0     0     0
    0     0     0     0     0     0
    0     0     0     0     1     0
    0     0     0     0     0     0
    0     1     0     1     0     0
    0     0     0     0     0     0
dfsArrayR =
    1     3     5     4     2     6
MATLAB implementation of relational algebra boolean matrix operations in inline functions
adjList =
    1     3     0     0
    2     1     3     4
    3     5     0     0
    4     3     0     0
    5     2     4     0
    6     4     5     0
nrowL =
    6
ncoll =
    4
treeR =
    0     0     1     0     0     0
    0     0     0     0     0     0
    0     0     0     0     1     0
    0     0     0     0     0     0
    0     1     0     1     0     0
    0     0     0     0     0     0
dfsArrayR =
    1     3     5     4     2     6

```

K.4 Depth First Search applied to reporting graph structures

Contents

- Graph operations on management reporting structures
- Set up relational algebra functionality
- Adjacency matrix of management structure graph with vertex labels
- Example - seven managers
- Adjacency lists of 'reports to' Relation
- Spanning trees using Depth First Search

Graph operations on management reporting structures

```

%.....
%   TreesManStructuresUnionIntersect.m
%   Forms that management reporting structures take when
%   basic graph operations are performed on the trees
%   Unions and intersections of graphs compute
%.....
clc
clear all
format compact

```

Set up relational algebra functionality

```
%.....
RelationalAlgebraBoolean
%.....
```

MATLAB implementation of relational algebra boolean matrix operations in inline functions

output file prefix

```
tgfFilePre='YHMMComb'
```

```
tgfFilePre =
YHMMComb
```

Adjacency matrix of management structure graph with vertex labels

Example - seven managers in hierarchy Both levels in one M0 MA MB M1 M2 M3 M4

```
HMM= [ 0 0 0 0 0 0 0 ; ...
        1 0 0 0 0 0 0 ; ...
        1 0 0 0 0 0 0 ; ...
        0 1 0 0 0 0 0 ; ...
        0 1 0 0 0 0 0 ; ...
        0 0 1 0 0 0 0 ; ...
        0 0 1 0 0 0 0 ]
vertexLabels= { ...
    'M_0','M_A','M_B','M_1','M_2','M_3','M_4' ...
}
% Output to yEd data file
isHomog=1
fileRS=horzcat(tgfFilePre,'ReportingStructure','_tgf')
tgfWrite(fileRS,HMM,isHomog,vertexLabels,{});
% Compute adjacency list from adjacency matrix
[AdjListHMM,nrowL,ncollL] = AdjacencyList(HMM)
```

```
HMM =
    0     0     0     0     0     0     0
    1     0     0     0     0     0     0
    1     0     0     0     0     0     0
    0     1     0     0     0     0     0
    0     1     0     0     0     0     0
    0     0     1     0     0     0     0
    0     0     1     0     0     0     0

vertexLabels =
    Columns 1 through 6
    'M_0'    'M_A'    'M_B'    'M_1'    'M_2'    'M_3'
    Column 7
    'M_4'
isHomog =
    1
fileRS =
YHMMCombReportingStructure.tgf
File YHMMCombReportingStructure.tgf opened
nRows =
```

```

    7
nCols =
    7
File YHMMCombReportingStructure.tgf closed
AdjListHMM =
    1    0
    2    1
    3    1
    4    2
    5    2
    6    3
    7    3
nrowL =
    7
ncoll =
    2

```

Example - seven managers

First level MA MB & M1 M2 M3 M4

```

HMM1= [0 0 0 0 0 0 ; ...
        0 0 0 0 0 0 ; ...
        1 0 0 0 0 0 ; ...
        1 0 0 0 0 0 ; ...
        0 1 0 0 0 0 ; ...
        0 1 0 0 0 0 ]
vertexLabels1= { ...
    'M_A','M_B','M_1','M_2','M_3','M_4' ...
}
isHomog=1
fileHM1=horzcat(tgfFilePre,'ReportingStrucLevel1','.tgf')
tgfWrite(fileHM1,HMM1,isHomog,vertexLabels1,{});
% Count number of edges - both directions - normal matrix operations
R1= ones(1,size(HMM1,2))
C1= ones(size(HMM1,2),1)
R_HMM1= R1*HMM1
R_HMM1_C=R_HMM1*C1
% report vertex outdegrees - row sums
InD_HMM1= R1*HMM1
% report vertex indegrees - column sums
OutD_HMM1=HMM1*C1
% not directed - Symmetry - element total = 0
T_symm= HMM1-HMM1'
Test=abs(R1*abs(T_symm*C1))
% Alternative
Test=sum(sum(T_symm))
% Paths in A2_3 - vertex degrees also on diagonal
A2=HMM1*HMM1
A3=HMM1*A2
A4=HMM1*A3
A2b=productR(HMM1,HMM1)
A3b=productR(A2b,HMM1)
% Second level MA & MB to M0
HMM2= [0 0 0 ; ...
        1 0 0 ; ...
        1 0 0 ]
vertexLabels2= { ...
    'M_0','M_A','M_B' ...
}
% Output yEd graph display data
isHomog=1
fileHM2=horzcat(tgfFilePre,'ReportingStrucLevel2','.tgf')
tgfWrite(fileHM2,HMM2,isHomog,vertexLabels2,{});
% Count number of edges - both directions - normal matrix operations
R1= ones(1,size(HMM2,2))
C1= ones(size(HMM2,2),1)

```

```

R_HMM2= R1*HMM2
R_HMM2_C=R_HMM2*C1
% report vertex outdegrees - row sums
InD_HMM2= R1*HMM2
% report vertex indegrees - column sums
OutD_HMM2=HMM2*C1
% not directed - Symmetry - element total = 0
T_symm= HMM2-HMM2'
Test=abs(R1*abs(T_symm*C1))
% Alternative
Test=sum(sum(T_symm))
% Paths in A2_3 - vertex degrees also on diagonal
A2=HMM2*HMM2
A3=HMM2*A2
A4=HMM2*A3
A2b=productR(HMM2,HMM2)
A3b=productR(A2b,HMM2)

```

```

HMM1 =
    0    0    0    0    0    0
    0    0    0    0    0    0
    1    0    0    0    0    0
    1    0    0    0    0    0
    0    1    0    0    0    0
    0    1    0    0    0    0
vertexLabels1 =
    'M_A'    'M_B'    'M_1'    'M_2'    'M_3'    'M_4'
isHomog =
    1
fileHM1 =
YHMMCombReportingStrucLevel1.tgf
File YHMMCombReportingStrucLevel1.tgf opened
nRows =
    6
nCols =
    6
File YHMMCombReportingStrucLevel1.tgf closed
R1 =
    1    1    1    1    1    1
C1 =
    1
    1
    1
    1
    1
    1
R_HMM1 =
    2    2    0    0    0    0
R_HMM1_C =
    4
InD_HMM1 =
    2    2    0    0    0    0
OutD_HMM1 =
    0
    0
    1
    1
    1
    1
T_symm =
    0    0    -1    -1    0    0
    0    0    0    0    -1    -1
    1    0    0    0    0    0
    1    0    0    0    0    0
    0    1    0    0    0    0
    0    1    0    0    0    0
Test =
    8

```

```

Test =
0
A2 =
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
A3 =
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
A4 =
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
A2b =
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
A3b =
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
HMM2 =
0 0 0
1 0 0
1 0 0
vertexLabels2 =
'M_O' 'M_A' 'M_B'
isHomog =
1
fileHMM2 =
YHMMCombReportingStrucLevel2.tgf
File YHMMCombReportingStrucLevel2.tgf opened
nRows =
3
nCols =
3
File YHMMCombReportingStrucLevel2.tgf closed
R1 =
1 1 1
C1 =
1
1
1
R_HMM2 =
2 0 0
R_HMM2_C =
2
InD_HMM2 =
2 0 0
OutD_HMM2 =
0
1
1
T_symm =
0 -1 -1

```

```

      1      0      0
      1      0      0
Test =
      4
Test =
      0
A2 =
      0      0      0
      0      0      0
      0      0      0
A3 =
      0      0      0
      0      0      0
      0      0      0
A4 =
      0      0      0
      0      0      0
      0      0      0
A2b =
      0      0      0
      0      0      0
      0      0      0
A3b =
      0      0      0
      0      0      0
      0      0      0

```

Adjacency lists of 'reports to' Relation

```

[AdjListHMM2,nrowL2,ncolL2] = AdjacencyList(HMM2)
[AdjListHMM1,nrowL1,ncolL1] = AdjacencyList(HMM1)
[AdjListHMM,nrowL0,ncolL0] = AdjacencyList(HMM)

```

```

AdjListHMM2 =
      1      0
      2      1
      3      1
nrowL2 =
      3
ncolL2 =
      2
AdjListHMM1 =
      1      0
      2      0
      3      1
      4      1
      5      2
      6      2
nrowL1 =
      6
ncolL1 =
      2
AdjListHMM =
      1      0
      2      1
      3      1
      4      2
      5      2
      6      3
      7      3
nrowL0 =
      7
ncolL0 =
      2

```

Spanning trees using Depth First Search

determine spanning trees with DFS of managment structures display adjacency lists and output to yEd data files

```
[treeHMM2,dfsArrayHMM2] = DepthFirstSearch(HMM2')
HMM2TreeAdj=AdjacencyList(treeHMM2)
fileTree2=horzcat(tgfFilePre,'Tree2DFS',''.tgf')
tgfWrite(fileTree2,treeHMM2,isHomog,vertexLabels2,{});
[treeHMM1,dfsArrayHMM1] = DepthFirstSearch(HMM1')
HMM1TreeAdj=AdjacencyList(treeHMM1)
fileTree1=horzcat(tgfFilePre,'Tree1DFS',''.tgf')
tgfWrite(fileTree1,treeHMM1,isHomog,vertexLabels1,{});
[treeHMM,dfsArrayHMM] = DepthFirstSearch(HMM')
HMMTreeAdj=AdjacencyList(treeHMM)
fileTree=horzcat(tgfFilePre,'TreeDFS',''.tgf')
tgfWrite(fileTree,treeHMM,isHomog,vertexLabels,{});
```

MATLAB implementation of relational algebra boolean matrix operations in inline functions

```
adjList =
    1     2     3
    2     0     0
    3     0     0

nrowL =
    3

ncoll =
    3

treeHMM2 =
    0     1     1
    0     0     0
    0     0     0

dfsArrayHMM2 =
    1     3     2
HMM2TreeAdj =
    1     2     3
    2     0     0
    3     0     0

fileTree2 =
YHMMCombTree2DFS.tgf
File YHMMCombTree2DFS.tgf opened
nRows =
    3
nCols =
    3
File YHMMCombTree2DFS.tgf closed
MATLAB implementation of relational algebra boolean matrix operations in inline functions
adjList =
    1     3     4
    2     5     6
    3     0     0
    4     0     0
    5     0     0
    6     0     0

nrowL =
    6

ncoll =
    3

treeHMM1 =
    0     0     1     1     0     0
    0     0     0     0     1     1
    0     0     0     0     0     0
    0     0     0     0     0     0
    0     0     0     0     0     0
    0     0     0     0     0     0

dfsArrayHMM1 =
    1     4     3     2     6     5
HMM1TreeAdj =
    1     3     4
    2     5     6
```

```

3      0      0
4      0      0
5      0      0
6      0      0
fileTree1 =
YHMMCombTree1DFS.tgf
File YHMMCombTree1DFS.tgf opened
nRows =
6
nCols =
6
File YHMMCombTree1DFS.tgf closed
MATLAB implementation of relational algebra boolean matrix operations in inline functions
adjList =
1      2      3
2      4      5
3      6      7
4      0      0
5      0      0
6      0      0
7      0      0
nrowL =
7
ncoll =
3
treeHMM =
0      1      1      0      0      0      0
0      0      0      1      1      0      0
0      0      0      0      0      1      1
0      0      0      0      0      0      0
0      0      0      0      0      0      0
0      0      0      0      0      0      0
0      0      0      0      0      0      0
dfsArrayHMM =
1      3      7      6      2      5      4
HMMTreeAdj =
1      2      3
2      4      5
3      6      7
4      0      0
5      0      0
6      0      0
7      0      0
fileTree =
YHMMCombTreeDFS.tgf
File YHMMCombTreeDFS.tgf opened
nRows =
7
nCols =
7
File YHMMCombTreeDFS.tgf closed

```

The resulting trees have adjacency matrices which are the transpose of the original structure adjacency matrices as shown in figure K.4.

K.5 Inserting sub management structures into larger structures

Adjacency matrices containing representations of management structures can be combined by a sub matrix insertion process to yield higher level complete representations in adjacency matrix format. Vertex labelling of input graphs referred to vertex labels of combined graphs are used.

The MATLAB code for the function to build up the vertex label sets and the total graph representation in incidence matrix format is listed below.

```

function [Rout] = incidenceInsert(vertexLabelsAll,vertexLabelsIn,Rin)
%.....
% Build up incidence matrix by union of columns at correct diagonal

```

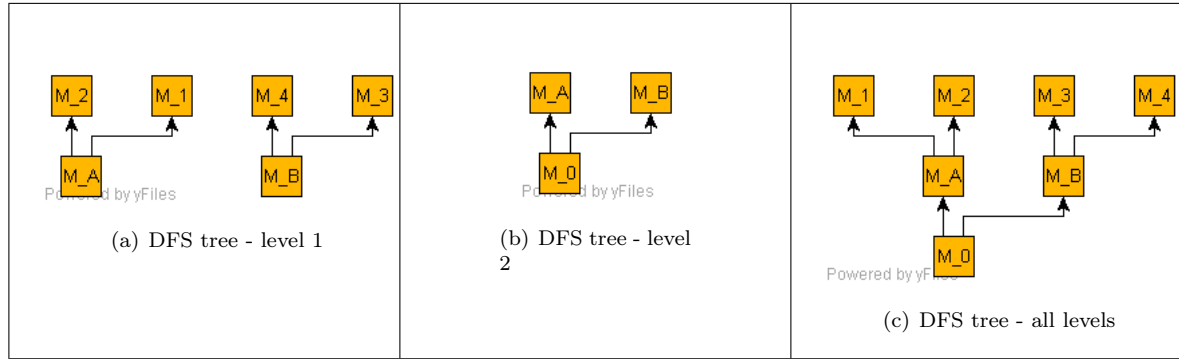



Figure K.2: Reporting or management structure - DFS tree representation: (a) Reporting structure DFS tree representation - level 1; (b) Reporting structure DFS tree representation - level 2; (c) Reporting structure DFS tree representation - all levels;

```
% to final matrix
% vertexLabelsAll - set of output vertex labels - cell array
% vertexLabelsIn - set of labels of inserted incidence matrix
% Rin - incidence matrix input
% Rout - incidence array output
% .....
% .....
RelationalAlgebraBoolean
% .....
nrowcol=size(vertexLabelsAll,2);
ivnum=size(vertexLabelsIn,2);
Rout=zeros(nrowcol,nrowcol);
for ivout=1:nrowcol
    for iv=1:ivnum
        % iv
        % char(vertexLabelsIn(iv))
        % ivout
        % char(vertexLabelsAll(ivout))
        if(char(vertexLabelsIn(iv))==char(vertexLabelsAll(ivout)))
            s2=size(Rin,2);
            Rout(ivout:ivout+s2-iv,ivout);
            Rin(iv:end,iv);
            Rinset=unionR(Rout(ivout:ivout+s2-iv,ivout),Rin(iv:end,iv));
            Rout(ivout:ivout+s2-iv,ivout)=Rinset;
        end
    end
end
end
```

Contents

- Graph operations on management reporting structures
- Set up relational algebra functionality
- Vertex labels and adjacency matrices of subgraphs
- Union of vertex label sets
- Use vertexLabels to build up incidence matrix

Graph operations on management reporting structures

```
% .....
% TreesManStructuresUnionIntersectPartB.m
% Forms that management reporting structures take when
% basic graph operations are performed on the trees
% Unions and intersections of graphs compute
% .....
clc
clear all
format compact
```

Set up relational algebra functionality

```
%.....
RelationalAlgebraBoolean
%.....
```

MATLAB implementation of relational algebra boolean matrix operations in inline functions

output file prefix

```
tgfFilePre='YHMMComb'
```

```
tgfFilePre =
YHMMComb
```

Vertex labels and adjacency matrices of subgraphs

First level MA MB & M1 M2 M3 M4

```
vertexLabels1= { ...
    'M_A','M_B','M_1','M_2','M_3','M_4' ...
}
HMM1= [0 0 0 0 0 0 ; ...
        0 0 0 0 0 0 ; ...
        1 0 0 0 0 0 ; ...
        1 0 0 0 0 0 ; ...
        0 1 0 0 0 0 ; ...
        0 1 0 0 0 0 ]
% Second level MA & MB to M0
vertexLabels2= { ...
    'M_0','M_A','M_B' ...
}
HMM2= [0 0 0 ; ...
        1 0 0 ; ...
        1 0 0 ]
```

```
vertexLabels1 =
    'M_A'    'M_B'    'M_1'    'M_2'    'M_3'    'M_4'
HMM1 =
    0     0     0     0     0     0
    0     0     0     0     0     0
    1     0     0     0     0     0
    1     0     0     0     0     0
    0     1     0     0     0     0
    0     1     0     0     0     0
vertexLabels2 =
    'M_0'    'M_A'    'M_B'
HMM2 =
    0     0     0
    1     0     0
    1     0     0
```

Union of vertex label sets

```
vertexLabelsAll={};
ivnum2=size(vertexLabels2,2)
for iv2=1:ivnum2
    vertexLabelsAll=addElementToSet(vertexLabels2(iv2),vertexLabelsAll);
end
ivnum1=size(vertexLabels1,2)
for iv1=1:ivnum1
    vertexLabelsAll=addElementToSet(vertexLabels1(iv1),vertexLabelsAll);
end
vertexLabels2
vertexLabels1
vertexLabelsAll
```

```
ivnum2 =
     3
ivnum1 =
     6
vertexLabels2 =
    'M_0'    'M_A'    'M_B'
vertexLabels1 =
    'M_A'    'M_B'    'M_1'    'M_2'    'M_3'    'M_4'
vertexLabelsAll =
    Columns 1 through 6
    'M_0'    'M_A'    'M_B'    'M_1'    'M_2'    'M_3'
    Column 7
    'M_4'
```

Use vertexLabels to build up incidence matrix

Insert incidence matrix

```
[Rout2] = incidenceInsert(vertexLabelsAll,vertexLabels2,HMM2)
[Rout1] = incidenceInsert(vertexLabelsAll,vertexLabels1,HMM1)
Rout=unionR(Rout1,Rout2)
```

MATLAB implementation of relational algebra boolean matrix operations in inline functions

```
Rout2 =
     0     0     0     0     0     0     0
     1     0     0     0     0     0     0
     1     0     0     0     0     0     0
     0     0     0     0     0     0     0
     0     0     0     0     0     0     0
     0     0     0     0     0     0     0
     0     0     0     0     0     0     0
```

MATLAB implementation of relational algebra boolean matrix operations in inline functions

```
Rout1 =
     0     0     0     0     0     0     0
     0     0     0     0     0     0     0
     0     0     0     0     0     0     0
     0     1     0     0     0     0     0
     0     1     0     0     0     0     0
     0     0     1     0     0     0     0
     0     0     1     0     0     0     0
```

```
Rout =
     0     0     0     0     0     0     0
     1     0     0     0     0     0     0
     1     0     0     0     0     0     0
     0     1     0     0     0     0     0
     0     1     0     0     0     0     0
     0     0     1     0     0     0     0
     0     0     1     0     0     0     0
```

K.6 Extracting sub graphs linked to selected nodes

The MATLAB algorithm below traverses the adjacency list and enters the edges linked to a selected into an adjacency matrix which has the size of the original graph.

As vertices on active edges are encountered they are marked as being active in the sub graph in a list of vertices to be retained as used by the sub graph. In the final step of the algorithm only adjacency matrix rows and columns of active vertices are collected in the adjacency matrix of the sub graph extracted.

Vertices which do not belong to the original vertices or unconnected vertices lead to the generation of an empty adjacency matrix.

MATLAB Test Example with output:

Contents

- Set up adjacency matrix and vertex labels
- Form adjacency list from adjacency matrix and output
- Extract subgraph adjacency matrices and output

```
%.....
% testAdjMatrixExtractRef.m
% Test extract adjacency matrix from adjacency list
%.....
clc
clear all
format compact
% Output filename prefix
tgfFilePre='EADL'
```

```
tgfFilePre =
EADL
```

Set up adjacency matrix and vertex labels

Graph adjacency matrix and vertex labels

```
HMM= [ 0 0 0 0 0 0 0 ; ...
       1 0 0 0 0 0 0 ; ...
       1 0 0 0 0 0 0 ; ...
       0 1 0 0 0 0 0 ; ...
       0 1 0 0 0 0 0 ; ...
       0 0 1 0 0 0 0 ; ...
       0 0 1 0 0 0 0 ]
vertexLabels= { ...
    'M_0','M_A','M_B','M_1','M_2','M_3','M_4' ...
}
```

```
HMM =
    0     0     0     0     0     0     0
    1     0     0     0     0     0     0
    1     0     0     0     0     0     0
    0     1     0     0     0     0     0
    0     1     0     0     0     0     0
    0     0     1     0     0     0     0
    0     0     1     0     0     0     0

vertexLabels =
Columns 1 through 6
'M_0'    'M_A'    'M_B'    'M_1'    'M_2'    'M_3'
Column 7
'M_4'
```

Form adjacency list from adjacency matrix and output

Form adjacency list from adjacency matrix

```
[adjListHMM,nrowL0,ncolL0] = AdjacencyList(HMM)
% .tgf file output for yEd display input
isHomog=1
fileTC=horzcat(tgfFilePre,'ManStructure',' .tgf')
% Output adjacency matrix for yEd
tgfWrite(fileTC,HMM,isHomog,vertexLabels,{});
```

```
adjListHMM =
     1     0
     2     1
     3     1
     4     2
     5     2
     6     3
     7     3
nrowL0 =
     7
ncolL0 =
     2
isHomog =
     1
fileTC =
EADLManStructure.tgf
File EADLManStructure.tgf opened
nRows =
     7
nCols =
     7
File EADLManStructure.tgf closed
```

Extract subgraph adjacency matrices and output

loop over vertices and extract sub graphs

```
[nRows,nCols]=size(HMM)
for nvertex=1:nRows
[adjMatrix,vactive] = adjListExtract(adjListHMM,nvertex)
vertexLabelsSubCol=vertexLabels(find(vactive))
% [adjMatrix,vactive] = adjListMultExtract(adjListHMM,nvertex)
isHomog=1
fileTC=horzcat(tgfFilePre,'ManStructureVertex',int2str(nvertex),' .tgf')
% output adjacency matrix
tgfWrite(fileTC,adjMatrix,isHomog,vertexLabelsSubCol,{});
[adjsubListHMM,nrowL,ncolL] = AdjacencyList(adjMatrix)
% [adjMatrixTest]=adjMatrixFromList(adjsubListHMM)
end
```

```
nRows =
     7
nCols =
     7
adjMatrix =
     0     0     0
     1     0     0
     1     0     0
vactive =
```

```

1      1      1      0      0      0      0
vertexLabelsSubCol =
'M_0'      'M_A'      'M_B'
isHomog =
1
fileTC =
EADLManStructureVertex1.tgf
File EADLManStructureVertex1.tgf opened
nRows =
3
nCols =
3
File EADLManStructureVertex1.tgf closed
adjsubListHMM =
1      0
2      1
3      1
nrowL =
3
ncoll =
2
adjMatrix =
0      0      0
1      0      0
1      0      0
vactive =
0      1      0      1      1      0      0
vertexLabelsSubCol =
'M_A'      'M_1'      'M_2'
isHomog =
1
fileTC =
EADLManStructureVertex2.tgf
File EADLManStructureVertex2.tgf opened
nRows =
3
nCols =
3
File EADLManStructureVertex2.tgf closed
adjsubListHMM =
1      0
2      1
3      1
nrowL =
3
ncoll =
2
adjMatrix =
0      0      0
1      0      0
1      0      0
vactive =
0      0      1      0      0      1      1
vertexLabelsSubCol =
'M_B'      'M_3'      'M_4'
isHomog =
1
fileTC =
EADLManStructureVertex3.tgf
File EADLManStructureVertex3.tgf opened
nRows =
3
nCols =
3
File EADLManStructureVertex3.tgf closed
adjsubListHMM =
1      0
2      1
3      1
nrowL =
3

```

```

ncoll =
    2
adjMatrix =
    []
vactive =
    0    0    0    0    0    0    0
vertexLabelsSubCol =
    Empty cell array: 1-by-0
isHomog =
    1
fileTC =
    EADLManStructureVertex4.tgf
File EADLManStructureVertex4.tgf opened
nRows =
    0
nCols =
    0
File EADLManStructureVertex4.tgf closed
adjsubListHMM =
    []
nrowL =
    0
ncoll =
    0
adjMatrix =
    []
vactive =
    0    0    0    0    0    0    0
vertexLabelsSubCol =
    Empty cell array: 1-by-0
isHomog =
    1
fileTC =
    EADLManStructureVertex5.tgf
File EADLManStructureVertex5.tgf opened
nRows =
    0
nCols =
    0
File EADLManStructureVertex5.tgf closed
adjsubListHMM =
    []
nrowL =
    0
ncoll =
    0
adjMatrix =
    []
vactive =
    0    0    0    0    0    0    0
vertexLabelsSubCol =
    Empty cell array: 1-by-0
isHomog =
    1
fileTC =
    EADLManStructureVertex6.tgf
File EADLManStructureVertex6.tgf opened
nRows =
    0
nCols =
    0
File EADLManStructureVertex6.tgf closed
adjsubListHMM =
    []
nrowL =
    0
ncoll =
    0
adjMatrix =
    []
vactive =

```

```

0 0 0 0 0 0 0
vertexLabelsSubCol =
    Empty cell array: 1-by-0
isHomog =
    1
fileTC =
    EADLManStructureVertex7.tgf
File EADLManStructureVertex7.tgf opened
nRows =
    0
nCols =
    0
File EADLManStructureVertex7.tgf closed
adjsubListHMM =
    []
nrowL =
    0
ncoll =
    0

```

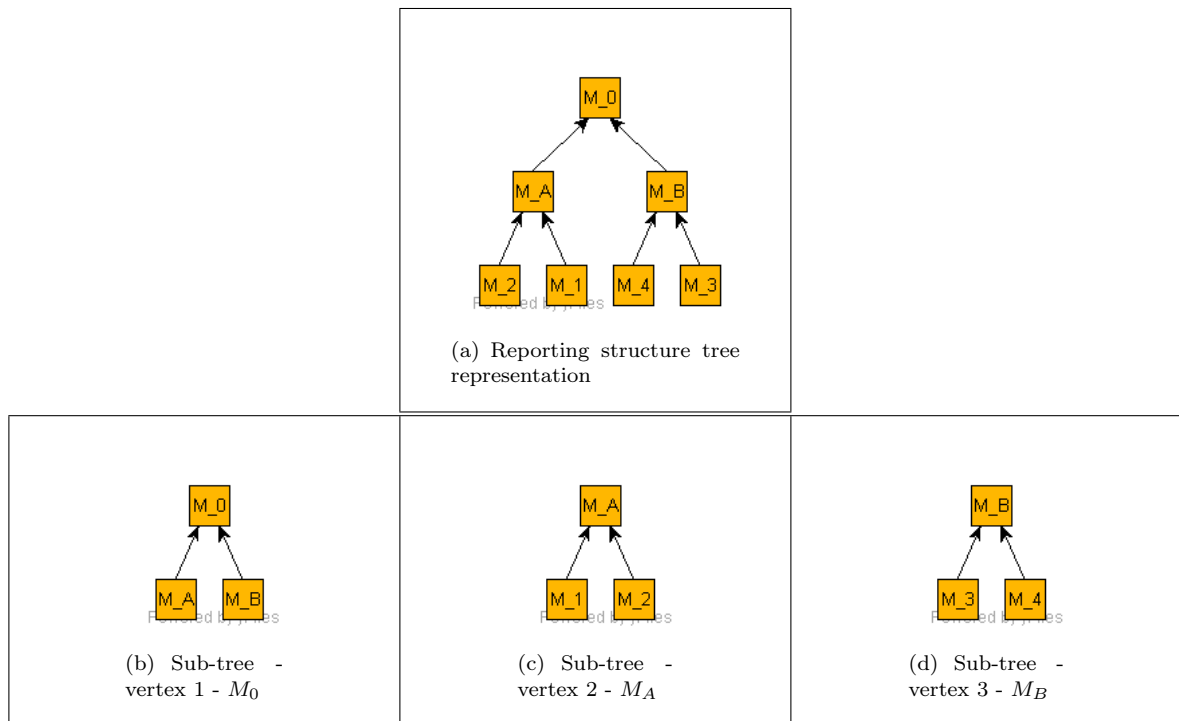


Figure K.3: Extracting sub-trees linked to vertices management structure - only non-null entries shown (a) Reporting structure tree representation (b) Sub-tree representation - vertex 1 - M_0 ; (c) Sub-tree representation - vertex 2 - M_A ; (d) Sub-tree representation - vertex 3 - M_B ;

The multiple vertex extraction logic which consists of a loop over previous function logic is shown in the MATLAB code below.

K.6.1 Graph adjacency matrix to list conversion

```

function [AdjList,nrowL,ncoll] = AdjacencyList(AdjMatrix)
% .....
% Generate adjacency list given adjacency matrix
% Adjacency list in matrix format - ignore 0 entries
% .....

```



```

AdjList=[];
nrowA=size(AdjMatrix,1);
ncolA=size(AdjMatrix,2);
for ir=1:nrowA
    irowL=ir;
    AdjList(irowL,1)=ir;
    icolL=1;
    for ic=1:ncolA
        if(AdjMatrix(ir,ic)==1)
            icolL=icolL+1;
            AdjList(irowL,icolL)=ic;
        end
    end
end
nrowL=size(AdjList,1);
ncolL=size(AdjList,2);

```

K.6.2 Graph adjacency list sub graph extraction

```

function [adjMatrix,vactive] = adjListMultExtract(adjList,nvertexList)
%.....
% function adjListMultExtract.m
% Extract all edges linked to a vertex and return
% adjMatrix - subgraph in adjacency matrix format
% vactive - Boolean list of active vertices
% can be converted to list format if required
% nvertexList - array of vertex numbers to process
%.....
% limits
nVertices=max(size(adjList,1),max(max(adjList)));
% set up blank adjacency matrix
adjMatrixInterim=zeros(nVertices);
vactive=zeros(1,nVertices);
for nvL=1:size(nvertexList,2)
    nvertex=nvertexList(nvL);
    if not((nvertex>nVertices))
% set limits to active vertex numbers
for nv=1:size(adjList,1)
    nEdgesRow=size(adjList(nv,:),2);
    for ne=2:nEdgesRow
% end vertex of edge
        mv=adjList(nv,ne);
% skip zero entries as well as entries not linked to selected vertex
        if not(mv==0)&&(mv==nvertex)
            adjMatrixInterim(nv,mv)=1;
% keep track of active vertex entries
            vactive(nv)=1;
            vactive(mv)=1;
        end
    end
end
end
end
% disp(['vactive: ',num2str(vactive)])
% adjMatrixInterim
% Add active columns to output matrix
adjCols=[];
for iv=1:nVertices
    if (vactive(iv)==1)
        adjCols=[adjCols,adjMatrixInterim(:,iv)];
    end
end
% Add active rows to output matrix
% adjCols
adjMatrix=[];
for iv=1:nVertices
    if (vactive(iv)==1)
        adjMatrix=[adjMatrix;adjCols(iv,:)];
    end
end

```

```

    end
end
end

```

```

function [adjMatrix,vactive] = adjListExtract(adjList,nvertex)
%.....
% Extract all edges linked to a vertex and return
% adjMatrix - subgraph in adjacency matrix format
% vactive - Boolean list of active vertices
% can be converted to list format if required
% nvertex - vertex number to process
%.....
% initialise output
adjMatrix=[];
nVertices=max(size(adjList,1),max(max(adjList)));
if not((nvertex>nVertices))
% set up blank adjacency matrix
adjMatrixInterim=zeros(nVertices);
vactive=zeros(1,nVertices);
% set limits to active vertex numbers
for nv=1:size(adjList,1)
    nEdgesRow=size(adjList(nv,:),2);
    for ne=2:nEdgesRow
% end vertex of edge
        mv=adjList(nv,ne);
% skip zero entries as well as entries not linked to selected vertex
        if not(mv==0)&& (mv==nvertex)
            adjMatrixInterim(nv,mv)=1;
% keep track of active vertex entries
            vactive(nv)=1;
            vactive(mv)=1;
        end
    end
end
% disp(['vactive: ',num2str(vactive)])
% adjMatrixInterim
% Add active columns to output matrix
adjCols=[];
for iv=1:nVertices
    if (vactive(iv)==1)
        adjCols=[adjCols,adjMatrixInterim(:,iv)];
    end
end
% Add active rows to output matrix
% adjCols
for iv=1:nVertices
    if (vactive(iv)==1)
        adjMatrix=[adjMatrix;adjCols(iv,:)];
    end
end
end
end

```

K.7 Management/ reporting structure - tree analysis examples

K.7.1 Basic example

The three level reporting structure used previously serves as the starting point for this example. Refer to figure K.4.

Contents

- Set up adjacency matrix and vertex labels
- Output data to file for yEd display
- Topological sorting and yEd data output
- Extract sub connectivity matrices
- Set up sub-matrix data and save to yEd display format

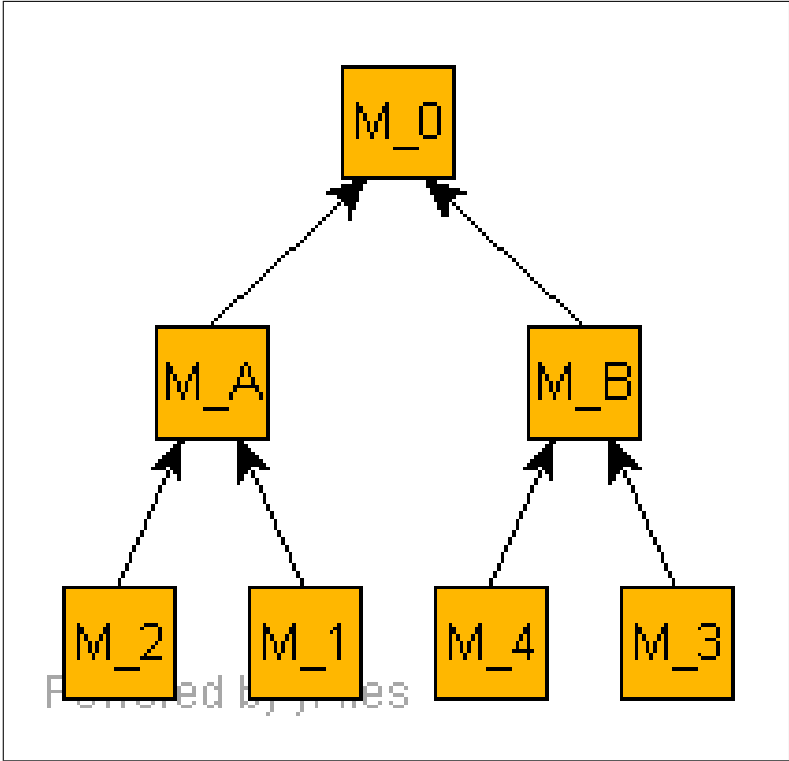


Figure K.4: Reporting structure adjacency matrix

```
%.....  
%   TreeSubConnectivity.m  
%.....  
%   Tree sub connectivity using BFS on Adjacency matrix  
%   extract columns per topoloogical sort level sequence  
%   remove zero rows from level per level adjacency matrices  
%   Sub adjacency matrices to be used for report roll up logic  
%.....  
clear all  
clc  
format compact
```

Set up adjacency matrix and vertex labels

```
tgfFilePre='TSCEx1'  
% Tree adjacency matrix  
TreeAdj=[0 0 0 0 0 0 0 ;...  
         1 0 0 0 0 0 0 ;...  
         1 0 0 0 0 0 0 ;...  
         0 1 0 0 0 0 0 ;...  
         0 1 0 0 0 0 0 ;...  
         0 0 1 0 0 0 0 ;...  
         0 0 1 0 0 0 0 ]  
vertexLabels= { ...  
    'M_0','M_A','M_B','M_1','M_2','M_3','M_4' ...  
}
```

```
tgfFilePre =  
TSCEx1  
TreeAdj =
```

```

0    0    0    0    0    0    0
1    0    0    0    0    0    0
1    0    0    0    0    0    0
0    1    0    0    0    0    0
0    1    0    0    0    0    0
0    0    1    0    0    0    0
0    0    1    0    0    0    0
vertexLabels =
Columns 1 through 6
'M_0'    'M_A'    'M_B'    'M_1'    'M_2'    'M_3'
Column 7
'M_4'
```

Output data to file for yEd display

```

isHomog=1
fileTC=horzcat(tgfFilePre,'TreeAdjConnectivity','.tgf')
tgfWrite(fileTC,TreeAdj,isHomog,vertexLabels,{});
```

```

isHomog =
1
fileTC =
TSCEx1TreeAdjConnectivity.tgf
File TSCEx1TreeAdjConnectivity.tgf opened
nRows =
7
nCols =
7
File TSCEx1TreeAdjConnectivity.tgf closed
```

Topological sorting and yEd data output

Topoogical sorting of tree vertices

```

TOut=TopolSort(TreeAdj')
levelLabels= { ...
    'L0','L1','L2' ...
}
% yEd display data for topological sort output
isHomog=0
fileTS=horzcat(tgfFilePre,'TreeAdjTopolSort','.tgf')
topolLabels=horzcat(vertexLabels,levelLabels)
tgfWrite(fileTS,TOut,isHomog,topolLabels,{});
```

```

MATLAB implementation of relational algebra boolean matrix operations in inline functions
T =
0    1    1    0    0    0    0
0    0    0    1    1    0    0
0    0    0    0    0    1    1
0    0    0    0    0    0    0
0    0    0    0    0    0    0
0    0    0    0    0    0    0
0    0    0    0    0    0    0
nVertex =
7
TOut =
1    0    0
0    1    0
```

```

0      1      0
0      0      1
0      0      1
0      0      1
0      0      1
levelLabels =
    'L0'    'L1'    'L2'
isHomog =
    0
fileTS =
TSCEx1TreeAdjTopolSort.tgf
topolLabels =
    Columns 1 through 6
    'M_0'    'M_A'    'M_B'    'M_1'    'M_2'    'M_3'
    Columns 7 through 10
    'M_4'    'L0'    'L1'    'L2'
File TSCEx1TreeAdjTopolSort.tgf opened
nRows =
    7
nCols =
    3
File TSCEx1TreeAdjTopolSort.tgf closed

```

Extract sub connectivity matrices

extract set of sub connectivity matrices from tree adjacency matrix according to topological sort matrix levels

```

[subAdj,subVactiveCol,subVactiveRow] ...
    = SubAdjMatricesExTopolSort(TreeAdj,TOut)
% Display results
% size(subAdj)
celldisp(subAdj)
celldisp(subVactiveCol)
celldisp(subVactiveRow)
% list id's of active vertices & submatrices & output for plot
nsub=size(subAdj,2)

```

```

nRow =
    7
nCol =
    3
subAdj =
    [2x1 double]    [4x2 double]
subVactiveCol =
    [1]    [1x3 double]
subVactiveRow =
    [7x1 double]    [7x1 double]
subAdj{1} =
    1
    1
subAdj{2} =
    1    0
    1    0
    0    1
    0    1
subVactiveCol{1} =
    1
subVactiveCol{2} =
    0    1    1
subVactiveRow{1} =
    0
    1
    1
    0

```

```

0
0
0
subVactiveRow{2} =
0
0
0
1
1
1
1
1
nsub =
2

```

Set up sub-matrix data and save to yEd display format

```

for is=1:nsub
vactiveCol=subVactiveCol{is}
vactiveRow=subVactiveRow{is}
vertexLabelsSubCol=vertexLabels(find(vactiveCol))
vertexLabelsSubRow=vertexLabels(find(vactiveRow))
adjMatrixR=subAdj{is}
isHomog=0;
fileOut=horzcat(tgfFilePre,'TreeSubConnec',num2str(is),'.tgf')
tgfWrite(fileOut,adjMatrixR,isHomog, ...
    horzcat(vertexLabelsSubRow,vertexLabelsSubCol),{ });
end

```

```

vactiveCol =
1
vactiveRow =
0
1
1
0
0
0
0
vertexLabelsSubCol =
'M_Q'
vertexLabelsSubRow =
'M_A' 'M_B'
adjMatrixR =
1
1
fileOut =
TSCEx1TreeSubConnec1.tgf
File TSCEx1TreeSubConnec1.tgf opened
nRows =
2
nCols =
1
File TSCEx1TreeSubConnec1.tgf closed
vactiveCol =
0 1 1
vactiveRow =
0
0
0
1
1
1
1
vertexLabelsSubCol =
'M_A' 'M_B'

```

```

vertexLabelsSubRow =
    'M_1'    'M_2'    'M_3'    'M_4'
adjMatrixR =
    1    0
    1    0
    0    1
    0    1
fileOut =
    TSCEx1TreeSubConnec2.tgf
File TSCEx1TreeSubConnec2.tgf opened
nRows =
    4
nCols =
    2
File TSCEx1TreeSubConnec2.tgf closed

```

Applying the algorithm shown in the MATLAB function code below with the example implementation leads to the two sub matrices displayed.

MATLAB code for function *SubAdjMatricesExTopolSort*.

```

function [subAdj,subVactiveCol,subVactiveRow] ...
    = SubAdjMatricesExTopolSort(TreeAdj,TOut)
% .....
% Extract set sub adjacency matrices form topological sort matrix
% given topol sort adjacency matrix as well as original tree
% structure adjacency matrix
% Input:
% TreeAdj – Tree adjacency matrix
% TOut – Topological sorting graph adjacency matrix
% Output:
% subAdj – Cell array of sub adjacency matrices
% subVactiveCol – Cell array of Boolean vectors of active vertices
%                included in subAdj – columns
% subVactiveRow – Cell array of Boolean vectors of active vertices
%                included in subAdj – rows
% .....
isubMat=0;
% size of topological sort adjacency matrix
% rows – vertices
% columns – sort levels / classes
nRow=size(TOut,1)
nCol=size(TOut,2)
% Loop over columns in topological sort graph adjacency matrix
for j=1:nCol
% list of vertices column list stores for extractions
    collist=[];
% boolean array of active vertices in position – columns
    vactiveCol=[];
% Loop over rows in topological sort adjacency matrix
    for i=1:nRow
        if(TOut(i,j)~= 0)
% add vertex to column list
            collist=[collist,i];
            vactiveCol(i)=1;
        end
    end
% Obtain set of active vertices for level op topological sort
    collist=unique(collist);
% Extract columns listed (if any)
    subMat=[];
    kCol=size(collist,2);
    if kCol > 0
        for k=1:kCol
            subMat=[subMat,TreeAdj(:,collist(k))] ;
        end
    end
% subMat
% number of rows in submatrix

```

```

    lRow=size(subMat,1);
% mark all vertices active for rows
    vactiveRow=ones(lRow,1);
    for lr=1:lRow
        if sum(subMat(lr,:))==0;
            vactiveRow(lr)=0;
        end
    end
% remove zero-rows from matrix subMat
    lr=1;
    while lr<=lRow
        if sum(subMat(lr,:))==0
            subMat(lr,:)=[];
            lRow=lRow-1;
            lr=0;
        end
        lr=lr+1;
    end % while
    if sum(subMat)==0
        subMat=[];
    end
% Store derived adjacency matrix in cell array
% subMat
% vactiveRow
% subAdj={} % - not correct
if (sum(size(subMat)))>0
    isubMat=isubMat+1;
    subAdj{isubMat}=subMat;
    subVactiveCol{isubMat}=vactiveCol;
    subVactiveRow{isubMat}=vactiveRow;
end
% next column
end % loop over topological sort adj matrix columns

```

The plot of the graph in this case with vertices L_0 , L_1 and L_2 assigned to the column vertex references is shown in figure K.5. L_0 , L_1 and L_2 represent the levels which each vertex in the graph is assigned to as shown in table K.1

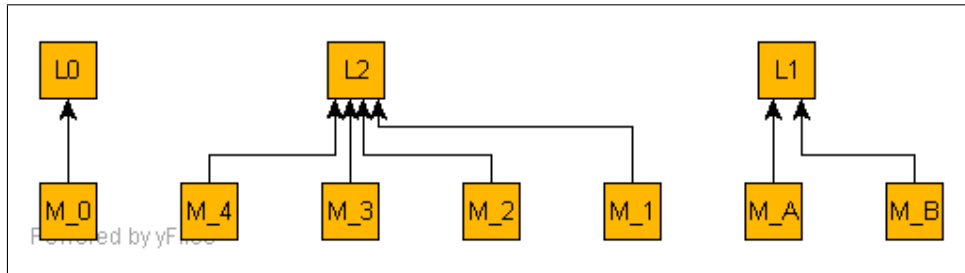


Figure K.5: Reporting structure adjacency matrix

The graphical representation of these adjacency matrices are shown in figure K.6.

The graphical representation of these adjacency matrices are shown in figure K.7.

The matrices computed above can be used to process the attributes assigned to lower levels in a 'roll up' process for management reporting at a higher level.

Level	Vertices in level of tree graph
L_0	M_0
L_1	M_A, M_B
L_2	M_1, M_2, M_3, M_4

Table K.1: Management reporting graph levels

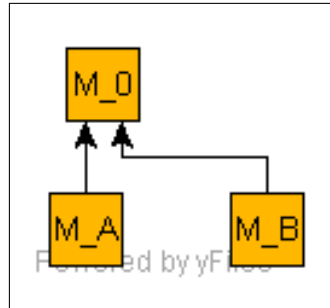


Figure K.6: Reporting structure adjacency matrix - Level 1 to Level 0

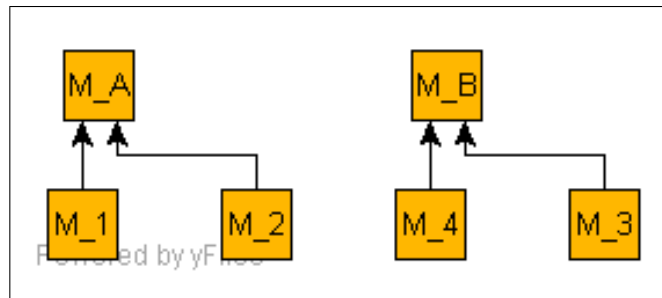


Figure K.7: Reporting structure adjacency matrix - Level2 to Level 1

K.7.2 Larger more realistic example

Contents

- Read graph data from yEd file
- Sort tree vertices by levels and output for yEd display
- Extract sub connectivity matrices
- Set up sub-matrix data and save to yEd display format

```
%.....
% TreeSubConnectivityExample2.m
% Larger Example from PEPE Example project
% Tree sub connectivity using BFS on Adjacency matrix
% Compute/extract sub adjacency matrices for each level of
% topological sort for use in report matrix data roll-up
%.....
clear all
clc
format compact
```

Read graph data from yEd file

```
tgfFile='PEPEXTsequenceT.tgf'
[vertexLabels,edgeLabels,TreeAdj] = TgfRead(tgfFile)
tgfFilePre='TSCEx2'
% Tree adjacency matrix
isHomog=1
fileTC=horzcat(tgfFilePre,'TreeAdjConnec','.tgf')
tgfWrite(fileTC,TreeAdj',isHomog,vertexLabels,{});
```

```
tgfFile =
PEPEXTsequenceT.tgf
File PEPEXTsequenceT.tgf opened
```



```
tgfFilePre =
TSCEx2
isHomog =
1
fileTC =
TSCEx2TreeAdjConvec.tgf
File TSCEx2TreeAdjConvec.tgf opened
nRows =
23
nCols =
23
File TSCEx2TreeAdjConvec.tgf closed
```

Topological sorting of tree vertices

```
    levelLabels(1:size(TOut,2)))
    tgfWrite(fileTS,TOut,isHomog,topolLabels,{});
```

```
T =
Columns 1 through 10
     0     1     1     1     1     0     1     1     1     0
     0     0     0     0     0     0     0     0     1     0
```



```

TOut =
    1     0     0     0     0     0
    0     1     0     0     0     0
    0     1     0     0     0     0
    0     0     1     0     0     0
    0     0     1     0     0     0
    0     0     0     1     0     0
    0     1     0     0     0     0
    0     0     1     0     0     0
    0     0     1     0     0     0
    0     0     0     1     0     0
    0     0     0     1     0     0
    0     0     0     0     1     0
    0     0     0     1     0     0
    0     0     1     0     0     0
    0     0     1     0     0     0
    0     0     0     1     0     0
    0     0     0     1     0     0
    0     0     0     0     1     0
    0     0     0     0     0     1
    0     0     0     1     0     0
    1     0     0     0     0     0
levelLabels =
    Columns 1 through 6
    'L0'    'L01'    'L02'    'L03'    'L04'    'L05'
    Columns 7 through 12
    'L06'    'L07'    'L08'    'L09'    'L10'    'L11'
    Column 13
    'L12'
isHomog =
    0
fileTS =
TSCEx2TreeAdjTopolSort.tgf
topolLabels =
    Columns 1 through 4
    [1x38 char]    [1x38 char]    [1x38 char]    [1x38 char]
    Columns 5 through 8
    [1x38 char]    [1x38 char]    [1x38 char]    [1x38 char]
    Columns 9 through 12
    [1x38 char]    [1x38 char]    [1x38 char]    [1x38 char]
    Columns 13 through 16
    [1x38 char]    [1x38 char]    [1x38 char]    [1x38 char]
    Columns 17 through 20
    [1x38 char]    [1x38 char]    [1x38 char]    [1x38 char]
    Columns 21 through 24
    [1x38 char]    [1x38 char]    [1x39 char]    'L0'
    Columns 25 through 29
    'L01'    'L02'    'L03'    'L04'    'L05'
File TSCEx2TreeAdjTopolSort.tgf opened
nRows =
    23
nCols =
    6
File TSCEx2TreeAdjTopolSort.tgf closed

```

Extract sub connectivity matrices

extract set of sub connectivity matrices from topological sort adjacency matrix

```

[subAdj,subVactiveCol,subVactiveRow] = ...
    SubAdjMatricesExTopolSort(TreeAdj,TOut);
% Display results
% size(subAdj)
celldisp(subAdj)
celldisp(subVactiveCol)

```

```
celldisp(subVactiveRow)
% Output graphs
nsub=size(subAdj,2)
```

```
nRow =
    23
nCol =
     6
subAdj{1} =
     1     1     1
subAdj{2} =
     1     1     1     1     0     0
     0     0     0     1     0     0
     1     1     0     0     0     1
     0     0     1     0     1     0
subAdj{3} =
     0     0     0     0     0     1     0
     0     0     0     0     0     1     0
     1     0     0     0     1     0     0
     0     1     0     0     0     0     0
     1     0     1     0     0     0     0
     0     0     0     0     0     0     1
     0     0     0     1     0     0     0
     0     0     0     0     0     1     0
     0     0     0     0     0     0     1
     0     0     0     0     1     0     0
subAdj{4} =
     0     1     0     0
     1     0     1     0
     1     0     0     0
     0     0     0     1
     0     1     0     0
     0     0     1     0
     0     0     0     1
subAdj{5} =
     1
     1
subVactiveCol{1} =
     0     1     1     0     0     0     1
subVactiveCol{2} =
Columns 1 through 10
     0     0     0     1     1     0     0     1     1     0
Columns 11 through 15
     0     0     0     1     1
subVactiveCol{3} =
Columns 1 through 10
     0     0     0     0     0     1     0     0     0     1
Columns 11 through 20
     1     0     1     0     0     1     1     0     0     0
Columns 21 through 22
     0     1
subVactiveCol{4} =
Columns 1 through 10
     0     0     0     0     0     0     0     0     0     0
Columns 11 through 20
     0     1     0     0     0     0     0     1     1     0
Column 21
     1
subVactiveCol{5} =
Columns 1 through 10
     0     0     0     0     0     0     0     0     0     0
Columns 11 through 20
     0     0     0     0     0     0     0     0     0     1
subVactiveRow{1} =
     1
     0
     0
     0
```


[illegible]

Set up sub-matrix data and save to yEd display format

```

for is=1:nsub
% for is=1:1
vactiveRow=subVactiveRow{is};
vertexLabelsSubRow=vertexLabels(find(vactiveRow))
celldisp(vertexLabelsSubRow)
vactiveCol=subVactiveCol{is};
vertexLabelsSubCol=vertexLabels(find(vactiveCol))
celldisp(vertexLabelsSubCol)
adjMatrixR=subAdj{is}
isHomog=0;
fileOut=horzcat(tgfFilePre,'TreeSubConne',num2str(is),'.tgf')
tgfWrite(fileOut,adjMatrixR,isHomog, ...
horzcat(vertexLabelsSubRow,vertexLabelsSubCol),{});
end

```



```

vertexLabelsSubRow =
    't01_Create architectural design      '
vertexLabelsSubRow{1} =
t01_Create architectural design
vertexLabelsSubCol =
    [1x38 char]    [1x38 char]    [1x38 char]
vertexLabelsSubCol{1} =
t02_Review architectural design
vertexLabelsSubCol{2} =
t03_Preliminary structural design
vertexLabelsSubCol{3} =
t07_Preliminary electrical design
adjMatrixR =
    1    1    1
fileOut =
TSCEx2TreeSubConnec1.tgf
File TSCEx2TreeSubConnec1.tgf opened
nRows =
    1
nCols =
    3
File TSCEx2TreeSubConnec1.tgf closed
vertexLabelsSubRow =
    [1x38 char]    [1x38 char]    [1x38 char]    [1x38 char]
vertexLabelsSubRow{1} =
t01_Create architectural design
vertexLabelsSubRow{2} =
t02_Review architectural design
vertexLabelsSubRow{3} =
t03_Preliminary structural design
vertexLabelsSubRow{4} =
t07_Preliminary electrical design
vertexLabelsSubCol =
    Columns 1 through 4
    [1x38 char]    [1x38 char]    [1x38 char]    [1x38 char]
    Columns 5 through 6
    [1x38 char]    [1x38 char]
vertexLabelsSubCol{1} =
t04_Create foundation drawings
vertexLabelsSubCol{2} =
t05_Create concrete layout drawings
vertexLabelsSubCol{3} =
t08_Create electrical drawings
vertexLabelsSubCol{4} =
t09_Finalize architectural drawings
vertexLabelsSubCol{5} =
t14_Finalize electrical design
vertexLabelsSubCol{6} =
t15_Finalize structural design
adjMatrixR =
    1    1    1    0    0
    0    0    0    1    0
    1    1    0    0    0    1
    0    0    1    0    1    0
fileOut =
TSCEx2TreeSubConnec2.tgf
File TSCEx2TreeSubConnec2.tgf opened
nRows =
    4
nCols =
    6
File TSCEx2TreeSubConnec2.tgf closed
vertexLabelsSubRow =
    Columns 1 through 4
    [1x38 char]    [1x38 char]    [1x38 char]    [1x38 char]
    Columns 5 through 8
    [1x38 char]    [1x38 char]    [1x38 char]    [1x38 char]
    Columns 9 through 10
    [1x38 char]    [1x38 char]
vertexLabelsSubRow{1} =
t01_Create architectural design
vertexLabelsSubRow{2} =
t02_Review architectural design

```

```

vertexLabelsSubRow{3} =
t03_Preliminary structural design
vertexLabelsSubRow{4} =
t04_Create foundation drawings
vertexLabelsSubRow{5} =
t05_Create concrete layout drawings
vertexLabelsSubRow{6} =
t07_Preliminary electrical design
vertexLabelsSubRow{7} =
t08_Create electrical drawings
vertexLabelsSubRow{8} =
t09_Finalize architectural drawings
vertexLabelsSubRow{9} =
t14_Finalize electrical design
vertexLabelsSubRow{10} =
t15_Finalize structural design
vertexLabelsSubCol =
  Columns 1 through 4
    [1x38 char] [1x38 char] [1x38 char] [1x38 char]
  Columns 5 through 7
    [1x38 char] [1x38 char] [1x38 char]
vertexLabelsSubCol{1} =
t06_Create reinforcement drawings
vertexLabelsSubCol{2} =
t10_Finalize foundation drawings
vertexLabelsSubCol{3} =
t11_Finalize concrete layout drawings
vertexLabelsSubCol{4} =
t13_Finalize electrical drawings
vertexLabelsSubCol{5} =
t16_Check structural design
vertexLabelsSubCol{6} =
t17_Check architectural drawings
vertexLabelsSubCol{7} =
t22_Check electrical design
adjMatrixR =
    0    0    0    0    0    1    0
    0    0    0    0    0    1    0
    1    0    0    0    1    0    0
    0    1    0    0    0    0    0
    1    0    1    0    0    0    0
    0    0    0    0    0    0    1
    0    0    0    1    0    0    0
    0    0    0    0    0    1    0
    0    0    0    0    0    0    1
    0    0    0    0    1    0    0
fileOut =
TSCEx2TreeSubConnec3.tgf
File TSCEx2TreeSubConnec3.tgf opened
nRows =
10
nCols =
7
File TSCEx2TreeSubConnec3.tgf closed
vertexLabelsSubRow =
  Columns 1 through 4
    [1x38 char] [1x38 char] [1x38 char] [1x38 char]
  Columns 5 through 7
    [1x38 char] [1x38 char] [1x38 char]
vertexLabelsSubRow{1} =
t04_Create foundation drawings
vertexLabelsSubRow{2} =
t05_Create concrete layout drawings
vertexLabelsSubRow{3} =
t06_Create reinforcement drawings
vertexLabelsSubRow{4} =
t08_Create electrical drawings
vertexLabelsSubRow{5} =
t10_Finalize foundation drawings
vertexLabelsSubRow{6} =
t11_Finalize concrete layout drawings

```

```

vertexLabelsSubRow{7} =
t13_Finalize electrical drawings
vertexLabelsSubCol =
    [1x38 char]    [1x38 char]    [1x38 char]
vertexLabelsSubCol{1} =
t12_Finalize reinforcement drawings
vertexLabelsSubCol{2} =
t18_Check foundation drawings
vertexLabelsSubCol{3} =
t19_Check concrete layout drawings
vertexLabelsSubCol{4} =
t21_Check electrical drawings
adjMatrixR =
    0    1    0    0
    1    0    1    0
    1    0    0    0
    0    0    0    1
    0    1    0    0
    0    0    1    0
    0    0    0    1
fileOut =
TSCEx2TreeSubConnec4.tgf
File TSCEx2TreeSubConnec4.tgf opened
nRows =
    7
nCols =
    4
File TSCEx2TreeSubConnec4.tgf closed
vertexLabelsSubRow =
    [1x38 char]    [1x38 char]
vertexLabelsSubRow{1} =
t06_Create reinforcement drawings
vertexLabelsSubRow{2} =
t12_Finalize reinforcement drawings
vertexLabelsSubCol =
    't20_Check reinforcement drawings    '
vertexLabelsSubCol{1} =
t20_Check reinforcement drawings
adjMatrixR =
    1
    1
fileOut =
TSCEx2TreeSubConnec5.tgf
File TSCEx2TreeSubConnec5.tgf opened
nRows =
    2
nCols =
    1
File TSCEx2TreeSubConnec5.tgf closed

```

```

function [vertexLabels,edgeLabels,R] = TgfRead(tgfFile)
%.....
%   tgfFile: File for input of data
%   vertexLabels: cell array with vertex label strings
%   edgeLabels: cell array with edge label strings
%   R adjacency matrix
%.....
% Input / Read .tgf file for relation for plotting with yEd
% Sample file
% 1 O
% 2 1
% 3 2
% 4 A
% 5 B
% 6 C
% 7 D
% 8 E
% #

```

Table K.2: Data file for tasks : PEPEXTsequenceT.tgf

1	Create architectural design	t01
2	Review architectural design	t02
3	Preliminary structural design	t03
4	Create foundation drawings	t04
5	Create concrete layout drawings	t05
6	Create reinforcement drawings	t06
7	Preliminary electrical design	t07
8	Create electrical drawings	t08
9	Finalize architectural drawings	t09
10	Finalize foundation drawings	t10
11	Finalize concrete layout drawings	t11
12	Finalize reinforcement drawings	t12
13	Finalize electrical drawings	t13
14	Finalize electrical design	t14
15	Finalize structural design	t15
16	Check structural design	t16
17	Check architectural drawings	t17
18	Check foundation drawings	t18
19	Check concrete layout drawings	t19
20	Check reinforcement drawings	t20
21	Check electrical drawings	t21
22	Check electrical design	t22
23	Tasks sequence with Tasks - Rule 1&2&3	
#	1	2
1	3	
1	4	
1	5	
1	7	
1	8	
1	9	
1	17	
2	9	
2	17	
3	4	
3	5	
3	6	
3	15	
3	16	
4	10	
4	18	
5	6	
5	11	
5	12	
5	19	
6	12	
6	20	
7	8	
7	14	
7	22	
8	13	
8	21	
9	17	
10	18	
11	19	
12	20	
13	21	
14	22	
15	16	

%	2	1	Edge1O
%	3	1	Edge2O
%	4	2	EdgeA1
%	5	2	EdgeB1
%	6	2	EdgeC1
%	7	3	EdgeD2
%	8	3	EdgeE2

```

% .....
% open file
fid=fopen(tgfFile,'r');
display(horzcat('File ',tgfFile,' opened'))
lineInputV='';
% vertices
vertexLabels={};
numVertices=0;
while not(lineInputV(1,1)=='#')
    lineInputV=fgetL(fid);
    if lineInputV(1,1)=='#'
        break
    end
    numVertices=numVertices+1;
    V=textscan(lineInputV,'%d %s','delimiter','\n');
%    celldisp(V)
    vertexLabels(numVertices)={' '};
    if(size(V{2},1) > 0)
        vertexLabels(numVertices)=V{2};
    end
end
% edges labels & adjacency matrix
edgeLabelsIn={};
% R=logical([;]);
% adjacency matrix to be square
R=logical(zeros(numVertices));
lineInputE='';
Rows=0;
Cols=0;
while not(lineInputE=='-1')
    lineInputE=fgetL(fid);
    if lineInputE=='-1'
        break
    end
    E=textscan(lineInputE,'%d %d %s','delimiter','\n');
    vertex1=double(E{1});
    if(vertex1>Rows) Rows=vertex1;end
    vertex2=double(E{2});
    if(vertex2>Cols) Cols=vertex2;end
    R(vertex1,vertex2)=1;
    edgeLabelsIn(vertex1,vertex2)={' '};
    if(size(E{3},1) > 0)
        edgeLabelsIn(vertex1,vertex2)=E{3};
    end
end
%Rows
%Cols
%sum(sum(R))
% store edgelabels in sequence of edges define in R
% row by row
edgeLabels={};
numEdges=0;
for ir=1:Rows
    for ic=1:Cols
        if(R(ir,ic)==1)
            numEdges=numEdges+1;
            edgeLabels(numEdges)=edgeLabelsIn(ir,ic);
        end
    end
end
end
%size(R)
%size(edgeLabels)
% close output file
fstatus=fclose(fid);
display(horzcat('File ',tgfFile,' closed'))

```

```

function [] = TgfWrite(tgfFile,R,isHomog,vertexLabels,edgeLabels)
% .....

```

```

% Output relation data file in directed graph data .tgf file format for use in
% yEd/yFiles software
% tgfFile: file name / Character string
% isHomogeneous: = 1 or 0 relation between one or two sets / bipartite graph
% R: Adjacency / boolean relation matrix / double array
% vertexLabels: vertex labels for plot / String cell array – required
% edgeLabels: edge labels for plot / String cell array – optional
% .....
% sample filename:
% tgfFile='TestFileA.tgf'
% Output .tgf file for relation for plotting with yEd
% Sample file contents:
% 1 O
% 2 1
% 3 2
% 4 A
% 5 B
% 6 C
% 7 D
% 8 E
% #
% 2 1 Edge1O
% 3 1 Edge2O
% 4 2 EdgeA1
% 5 2 EdgeB1
% 6 2 EdgeC1
% 7 3 EdgeD2
% 8 3 EdgeE2
% Sample vertexLabels:
% vertexLabels={'O','1','2','A','B','C','D','E'}
% Sample edgelabels:
% edgeLabels={'Edge1O', ...
% 'Edge2O', ...
% 'EdgeA1', ...
% 'EdgeB1', ...
% 'EdgeC1', ...
% 'EdgeD2', ...
% 'EdgeE2'}
% Sample relation boolean matrix / adjacency matrix for directed graph:
% R=logical([0 1 1 0 0 0 0 0; ...
% 0 0 0 1 1 1 0 0; ...
% 0 0 0 0 0 0 0 1; ...
% 0 0 0 0 1 1 0 0; ...
% 0 0 1 1 0 0 0 0; ...
% 1 1 0 0 0 0 0 0; ...
% 1 0 0 0 1 1 1 1])
% .....%
% loop over entries and output data
% list vertices
fid=fopen(tgfFile,'w');
display (horzcat('File ',tgfFile,' opened'))
for i=1:size(vertexLabels,2)
    % output only non-zero length labels with
    if size(vertexLabels{i},2)>0
        fprintf(fid,'%u %s \n',i,vertexLabels{i});
    end
end
fprintf(fid,'%s \n','#');
% homogeneous relation
% loop over relation boolean matrix to output edges
iedge=0;
nRows=size(R,1)
nCols=size(R,2)
for j=1:nRows
    for k=1:nCols
        if(R(j,k) == true )
            iedge=iedge+1;
            label='';
            if(iedge <= size(edgeLabels,2));
                label=edgeLabels{iedge};
            end
        end
    end
end

```

```

        jpos=j;
        kpos=k;
        if (not (isHomog==1))
            kpos=k+nRows;
        end
        fprintf(fid, '%u %u %s \n', jpos, kpos, label);
    end
end
end
% close output file
fstatus=fclose(fid);
display (horzcat('File ',tgfFile, ' closed'))

```

Topological sort sequence - note header vertex in graph added to level L_0 in this case shown in figure K.8.

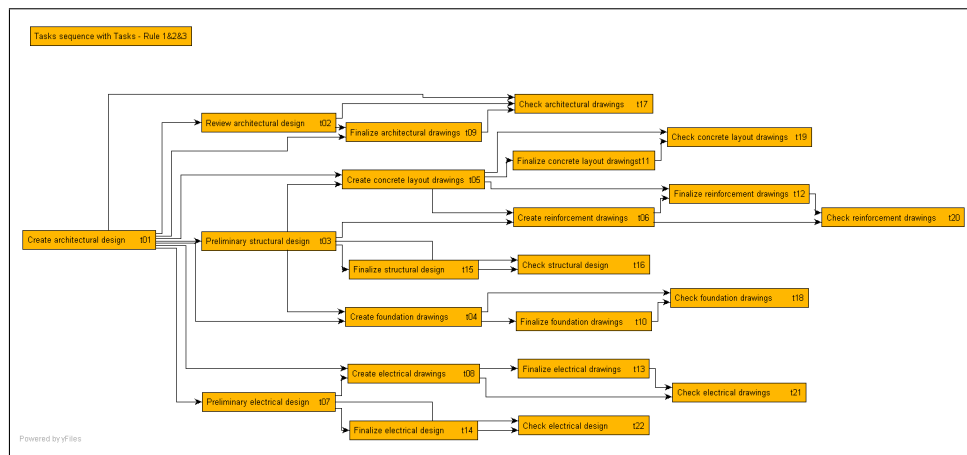


Figure K.8: *Reporting structure adjacency matrix*

Sub adjacency graphs with abbreviated adjacency matrices in this case are listed in the MATLAB output display.

K.8 Determining connectivity of vertices in graphs e.g. to determine tree vertex links for roll up of reports

```

function [adjMatrix,vactive] = adjListExtract(adjList,nvertex)
% .....
% Extract all edges linked to a vertex and return
% adjMatrix - subgraph in adjacency matrix format
% vactive - Boolean list of active vertices
% can be converted to list format if required
% nvertex - vertex number to process
% .....
% initialise output
adjMatrix=[];
nVertices=max(size(adjList,1),max(max(adjList)));
if not((nvertex>nVertices))
% set up blank adjacency matrix
adjMatrixInterim=zeros(nVertices);
vactive=zeros(1,nVertices);
% set limits to active vertex numbers
for nv=1:size(adjList,1)
    nEdgesRow=size(adjList(nv,:),2);
    for ne=2:nEdgesRow

```

```

% end vertex of edge
    mv=adjList(nv,ne);
% skip zero entries as well as entries not linked to selected vertex
    if not(mv==0)&& (mv==nvertex)
        adjMatrixInterim(nv,mv)=1;
% keep track of active vertex entries
        vactive(nv)=1;
        vactive(mv)=1;
    end
end
end
% disp(['vactive: ',num2str(vactive)])
% adjMatrixInterim
% Add active columns to output matrix
    adjCols=[];
for iv=1:nVertices
    if (vactive(iv)==1)
        adjCols=[adjCols,adjMatrixInterim(:,iv)];
    end
end
% Add active rows to output matrix
% adjCols
for iv=1:nVertices
    if (vactive(iv)==1)
        adjMatrix=[adjMatrix;adjCols(iv,:)];
    end
end
end
end

```

```

function [AdjList,nrowL,ncolL] = AdjacencyList(AdjMatrix)
% .....
% Generate adjacency list given adjacency matrix
% Adjacency list in matrix format – ignore 0 entries
% .....
AdjList=[];
nrowA=size(AdjMatrix,1);
ncolA=size(AdjMatrix,2);
for ir=1:nrowA
    irowL=ir;
    AdjList(irowL,1)=ir;
    icolL=1;
    for ic=1:ncolA
        if (AdjMatrix(ir,ic)==1)
            icolL=icolL+1;
            AdjList(irowL,icolL)=ic;
        end
    end
end
end
nrowL=size(AdjList,1);
ncolL=size(AdjList,2);

```

The logic for extracting the connectivity of a given vertex in a graph is given in the MATLAB code below. Edges from vertices leading into a given vertex is determined and the graph determined.

K.9 Report data roll up using adjacency matrices

K.9.1 Theoretical Example using topological sorting and sub matrix extraction

Contents

- Reported symbolic values
- Define reporting structure graph and vertex labels
- Topological sort to determine roll up levels
- Compute report roll up sets


```
%.....
% ReportResultsRollUpTopol.m
% Report results roll up using topological sorting
%.....
clear all
clc
format compact
```

Reported symbolic values

vectors of reported symbolic values

```
syms x1 x2 x3 x4 y1 y2 y3 y4 z1 z2 z3 z4 real
w1=[x1 x2 x3 x4]'
w2=[y1 y2 y3 y4]'
w3=[z1 z2 z3 z4]'
% matrix of vectors of reported values
w=[w1,w2,w3]
```

```
w1 =
    x1
    x2
    x3
    x4
w2 =
    y1
    y2
    y3
    y4
w3 =
    z1
    z2
    z3
    z4
w =
[ x1, y1, z1]
[ x2, y2, z2]
[ x3, y3, z3]
[ x4, y4, z4]
```

Define reporting structure graph and vertex labels

Adjacency matrix of graph of reporting structure

```
HMM= [ 0 0 0 0 0 0 0 ; ...
       1 0 0 0 0 0 0 ; ...
       1 0 0 0 0 0 0 ; ...
       0 1 0 0 0 0 0 ; ...
       0 1 0 0 0 0 0 ; ...
       0 0 1 0 0 0 0 ; ...
       0 0 1 0 0 0 0 ]
vertexLabels= { ...
    'M_0','M_A','M_B','M_1','M_2','M_3','M_4' ...
}
```

```

HMM =
    0     0     0     0     0     0     0
    1     0     0     0     0     0     0
    1     0     0     0     0     0     0
    0     1     0     0     0     0     0
    0     1     0     0     0     0     0
    0     0     1     0     0     0     0
    0     0     1     0     0     0     0

vertexLabels =
  Columns 1 through 6
  'M_0'    'M_A'    'M_B'    'M_1'    'M_2'    'M_3'
  Column 7
  'M_4'

```

Topological sort to determine roll up levels

Topological Sort on Transpose of Adjacency Matrix

```

TOut=TopolSort(HMM')
[subAdj,subVactiveCol,subVactiveRow] ...
    = SubAdjMatricesExTopolSort(HMM,TOut)
% Display results
% size(subAdj)
celldisp(subAdj)
celldisp(subVactiveCol)
celldisp(subVactiveRow)
% list id's of active vertices & submatrices & output for plot
nsub=size(subAdj,2)

```

```

MATLAB implementation of relational algebra boolean matrix operations in inline functions
T =
    0     1     1     0     0     0     0
    0     0     0     1     1     0     0
    0     0     0     0     0     1     1
    0     0     0     0     0     0     0
    0     0     0     0     0     0     0
    0     0     0     0     0     0     0
    0     0     0     0     0     0     0

nVertex =
    7
TOut =
    1     0     0
    0     1     0
    0     1     0
    0     0     1
    0     0     1
    0     0     1
    0     0     1

nRow =
    7
nCol =
    3
subAdj =
    [2x1 double]    [4x2 double]
subVactiveCol =
    [1]    [1x3 double]
subVactiveRow =
    [7x1 double]    [7x1 double]
subAdj{1} =
    1
    1
subAdj{2} =
    1     0
    1     0
    0     1

```

```

    0    1
subVactiveCol{1} =
    1
subVactiveCol{2} =
    0    1    1
subVactiveRow{1} =
    0
    1
    1
    0
    0
    0
    0
subVactiveRow{2} =
    0
    0
    0
    1
    1
    1
    1
    1
nsub =
    2

```

Compute report roll up sets

Lowest Level rollup

```

subAdjUse=subAdj{nsub}
wL1=(w'*subAdjUse)'
% Second Level rollup
subAdjUse=subAdj{1}
wL2=(wL1'*subAdjUse)'

```

```

subAdjUse =
    1    0
    1    0
    0    1
    0    1
wL1 =
[ x1+x2, y1+y2, z1+z2]
[ x3+x4, y3+y4, z3+z4]
subAdjUse =
    1
    1
wL2 =
[ x1+x2+x3+x4, y1+y2+y3+y4, z1+z2+z3+z4]

```

K.9.2 Theoretical Example - Using graph adjacency list processing

Contents

- Reported symbolic values
- Define reporting structure graph and vertex labels
- Convert to adjacency list format -
- Extract sub graphs from adjacency list and compute roll up
- Process top level

```

% .....
% ReportResultsRollUp.m
% Report results roll up

```

```
%.....
clear all
clc
format compact
```

Reported symbolic values

vectors of reported symbolic values

```
syms x1 x2 x3 x4 y1 y2 y3 y4 z1 z2 z3 z4 real
w1=[x1 x2 x3 x4]',
w2=[y1 y2 y3 y4]',
w3=[z1 z2 z3 z4]',
w=[w1,w2,w3]
```

```
w1 =
    x1
    x2
    x3
    x4
w2 =
    y1
    y2
    y3
    y4
w3 =
    z1
    z2
    z3
    z4
w =
[ x1, y1, z1]
[ x2, y2, z2]
[ x3, y3, z3]
[ x4, y4, z4]
```

Define reporting structure graph and vertex labels

Adjacency matrix of graph of reporting structure

```
HMM= [ 0 0 0 0 0 0 0 ; ...
        1 0 0 0 0 0 0 ; ...
        1 0 0 0 0 0 0 ; ...
        0 1 0 0 0 0 0 ; ...
        0 1 0 0 0 0 0 ; ...
        0 0 1 0 0 0 0 ; ...
        0 0 1 0 0 0 0 ]
vertexLabels= { ...
    'M_0','M_A','M_B','M_1','M_2','M_3','M_4' ...
}
```

```
HMM =
     0     0     0     0     0     0     0
     1     0     0     0     0     0     0
     1     0     0     0     0     0     0
     0     1     0     0     0     0     0
     0     1     0     0     0     0     0
     0     0     1     0     0     0     0
```

```

    0    0    1    0    0    0    0
vertexLabels =
  Columns 1 through 6
    'M_0'    'M_A'    'M_B'    'M_1'    'M_2'    'M_3'
  Column 7
    'M_4'
```

Convert to adjacency list format -

Adjacency list format

```
[adjList,nrowL,ncolL] = AdjacencyList(HMM)
```

```

adjList =
     1     0
     2     1
     3     1
     4     2
     5     2
     6     3
     7     3
nrowL =
     7
ncolL =
     2
```

Extract sub graphs from adjacency list and compute roll up

Process second & third levels Second level

```

nvertex=2
[adjMatrix,vactive] = adjListExtract(adjList,nvertex)
%fileOut=horzcat(tgfFilePre,'TreeSubConnec',num2str(nvertex),'.tgf')
vertexLabelsOut=vertexLabels(find(vactive))
% tgfWrite(fileOut,adjMatrixR,isHomog,vertexLabelsOut,{});
% convert sub graph to adjacency list format
[adjsubList,nrowS,ncolS] = AdjacencyList(adjMatrix)
% compute roll up
wA=w(1:2,:)
wAOutA=wA'*adjMatrix(2:end,1:1)
% Third level
nvertex=3
[adjMatrix,vactive] = adjListExtract(adjList,nvertex)
%fileOut=horzcat(tgfFilePre,'TreeSubConnec',num2str(nvertex),'.tgf')
vertexLabelsOut=vertexLabels(find(vactive))
% tgfWrite(fileOut,adjMatrixR,isHomog,vertexLabelsOut,{});
% convert sub graph to adjacency list format
[adjsubList,nrowS,ncolS] = AdjacencyList(adjMatrix)
% compute roll up
wB=w(3:4,:)
wAOutB=wB'*adjMatrix(2:end,1:1)
```

```

nvertex =
     2
adjMatrix =
     0     0     0
     1     0     0
```

```

    1    0    0
vactive =
    0    1    0    1    1    0    0
vertexLabelsOut =
    'M_A'    'M_1'    'M_2'
adjsubList =
    1    0
    2    1
    3    1
nrowS =
    3
ncolS =
    2
wA =
[ x1, y1, z1]
[ x2, y2, z2]
wAOutA =
    x1+x2
    y1+y2
    z1+z2
nvertex =
    3
adjMatrix =
    0    0    0
    1    0    0
    1    0    0
vactive =
    0    0    1    0    0    1    1
vertexLabelsOut =
    'M_B'    'M_3'    'M_4'
adjsubList =
    1    0
    2    1
    3    1
nrowS =
    3
ncolS =
    2
wB =
[ x3, y3, z3]
[ x4, y4, z4]
wAOutB =
    x3+x4
    y3+y4
    z3+z4

```

Process top level

```

nvertex=1
[adjMatrix,vactive] = adjListExtract(adjList,nvertex)
%fileOut=horzcat(tgfFilePre,'TreeSubConnec',num2str(nvertex),'.tgf')
vertexLabelsOut=vertexLabels(find(vactive))
% tgfWrite(fileOut,adjMatrixR,isHomog,vertexLabelsOut,{});
% convert sub graph to adjacency list format
[adjsubList,nrowS,ncolS] = AdjacencyList(adjMatrix)
w0=[wAOutA,wAOutB]
% note transpose complete on previous level
wAOut0=(w0*adjMatrix(2:end,1:1))'

```

```

nvertex =
    1
adjMatrix =
    0    0    0
    1    0    0
    1    0    0

```

```

vactive =
    1    1    1    0    0    0    0
vertexLabelsOut =
    'M_0'    'M_A'    'M_B'
adjsubList =
    1    0
    2    1
    3    1
nrowS =
    3
ncolS =
    2
w0 =
[ x1+x2, x3+x4]
[ y1+y2, y3+y4]
[ z1+z2, z3+z4]
wAOut0 =
[ x1+x2+x3+x4, y1+y2+y3+y4, z1+z2+z3+z4]

```

K.9.3 Larger example with numerical values

Contents

- Read in graph data from yEd .tgf file
- Topological sorting of graph and set up level labels
- Extract sub adjacency matrices and display
- Set up reporting weight and accumulation strings
- Roll up reporting string data

```

%.....
% ReportResultsRollUpTopolEx2.m
% Report results roll up
%.....
clear all
clc
format compact

```

Read in graph data from yEd .tgf file

graph of reporting structure - read ex data file

```

format compact
tgfFile='PEPEXTsequenceT.tgf'
[vertexLabels,edgeLabels,TreeAdj] = TgfRead(tgfFile)
% Output graph for yEd display - not used
%tgfFilePre='TSCEX2'
% Tree adjacency matrix
%isHomog=1
%fileTC=horzcat(tgfFilePre,'TreeAdjConne','T.tgf')
%tgfWrite(fileTC,TreeAdj,isHomog,vertexLabels,{});
% scratch last row & column of adjacency matrix
% contains header of graph - not necessary - no edge links vertex
% TreeAdj=TreeAdj(1:size(TreeAdj,1)-1,1:(size(TreeAdj,2)-1))
% Topological sorting of tree vertices

```

```

tgfFile =
PEPEXTsequenceT.tgf
File PEPEXTsequenceT.tgf opened
File PEPEXTsequenceT.tgf closed
vertexLabels =
Columns 1 through 4
    [1x38 char]    [1x38 char]    [1x38 char]    [1x38 char]

```



```

'L0'      'L01'      'L02'      'L03'      'L04'      'L05'
Columns 7 through 12
'L06'      'L07'      'L08'      'L09'      'L10'      'L11'
Column 13
'L12'

```

Extract sub adjacency matrices and display

```

[subAdj,subVactiveCol,subVactiveRow] ...
    = SubAdjMatricesExTopolSort(TreeAdj,TOut)
% Display results
% size(subAdj)
celldisp(subAdj)
celldisp(subVactiveCol)
celldisp(subVactiveRow)
% Production units type p01,p02,p03,p04
% Hours worked on tasks
% Persons:
% 'Client          p01', ...
% 'Architect       p02', ...
% 'Structural Engineer p03', ...
% 'Electrical Engineer p04', ...
% 'Draftsman       p05', ...
% 'Checking Engineer p06',

```

```

nRow =
    23
nCol =
     6
subAdj =
    Columns 1 through 3
    [1x3 double]    [4x6 double]    [10x7 double]
    Columns 4 through 5
    [7x4 double]    [2x1 double]
subVactiveCol =
    Columns 1 through 3
    [1x7 double]    [1x15 double]    [1x22 double]
    Columns 4 through 5
    [1x21 double]    [1x20 double]
subVactiveRow =
    Columns 1 through 3
    [23x1 double]    [23x1 double]    [23x1 double]
    Columns 4 through 5
    [23x1 double]    [23x1 double]
subAdj{1} =
     1     1     1
subAdj{2} =
     1     1     1     1     0     0
     0     0     0     1     0     0
     1     1     0     0     0     1
     0     0     1     0     1     0
subAdj{3} =
     0     0     0     0     0     1     0
     0     0     0     0     0     1     0
     1     0     0     0     1     0     0
     0     1     0     0     0     0     0
     1     0     1     0     0     0     0
     0     0     0     0     0     0     1
     0     0     0     1     0     0     0
     0     0     0     0     0     1     0
     0     0     0     0     0     0     1
     0     0     0     0     1     0     0
subAdj{4} =
     0     1     0     0
     1     0     1     0

```


0
0
0
0
0
0
0
0
0
0
0
0
0
subVactiveRow{3} =
1
1
1
1
1
0
1
1
1
0
0
0
0
1
1
0
0
0
0
0
0
0
subVactiveRow{4} =
0
0
0
1
1
1
0
1
0
1
1
0
1
0
0
0
0
0
0
0
0
0
subVactiveRow{5} =
0
0
0
0
0
1
0
0
0
0
0
0

```

1
0
0
0
0
0
0
0
0
0
0
0
0

```

Set up reporting weight and accumulation strings

```

nW=6
w=[ ...
1 1 0 0 0 0; ...% 'Create architectural design      t01', ...
1 1 0 0 0 0; ...% 'Review architectural design      t02', ...
0 0 1 0 0 0; ...% 'Preliminary structural design    t03', ...
0 0 0 1 0; ...% 'Create foundation drawings          t04', ...
0 0 0 0 1 0; ...% 'Create concrete layout drawings  t05', ...
0 0 0 0 1 0; ...% 'Create reinforcement drawings    t06', ...
0 0 0 1 0 0; ...% 'Preliminary electrical design    t07', ...
0 0 0 0 1 0; ...% 'Create electrical drawings       t08', ...
0 1 0 0 0 0; ...% 'Finalize architectural drawings  t09', ...
0 0 0 0 1 0; ...% 'Finalize foundation drawings     t10', ...
0 0 0 0 1 0; ...% 'Finalize concrete layout drawings t11', ...
0 0 0 0 1 0; ...% 'Finalize reinforcement drawings  t12', ...
0 0 0 0 1 0; ...% 'Finalize electrical drawings     t13', ...
0 0 0 1 0 0; ...% 'Finalize electrical design       t14', ...
0 0 1 0 0 0; ...% 'Finalize structural design       t15', ...
0 0 0 0 0 1; ...% 'Check structural design          t16', ...
0 0 0 0 0 1; ...% 'Check architectural drawings     t17', ...
0 0 0 0 0 1; ...% 'Check foundation drawings        t18', ...
0 0 0 0 0 1; ...% 'Check concrete layout drawings  t19', ...
0 0 0 0 0 1; ...% 'Check reinforcement drawings    t20', ...
0 0 0 0 0 1; ...% 'Check electrical drawings       t21', ...
0 0 0 0 0 1; ...% 'Check electrical design         t22' ...
]
wAccumulate=zeros((size(TreeAdj,1)-1),nW)
% number of submatrices
nsub=size(subAdj,2)

```

```

nW =
6
w =
1 1 0 0 0 0
1 1 0 0 0 0
0 0 1 0 0 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 1 0 0
0 0 0 0 1 0
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 1 0 0
0 0 1 0 0 0
0 0 0 0 0 1
0 0 0 0 0 1

```

	0	0	0	0	0	1
	0	0	0	0	0	1
	0	0	0	0	0	1
	0	0	0	0	0	1
wAccumulate =						
	0	0	0	0	0	0
	0	0	0	0	0	0
	0	0	0	0	0	0
	0	0	0	0	0	0
	0	0	0	0	0	0
	0	0	0	0	0	0
	0	0	0	0	0	0
	0	0	0	0	0	0
	0	0	0	0	0	0
	0	0	0	0	0	0
	0	0	0	0	0	0
	0	0	0	0	0	0
	0	0	0	0	0	0
	0	0	0	0	0	0
	0	0	0	0	0	0
	0	0	0	0	0	0
	0	0	0	0	0	0
	0	0	0	0	0	0
	0	0	0	0	0	0
	0	0	0	0	0	0
	0	0	0	0	0	0
nsub =						
	5					

Roll up reporting string data

```

for is=1:nsub
% Roll Up from Low to High
% adjacency matrix
subAdjUse=subAdj{is}
% active rows
vactiveRow=subVactiveRow{is};
% active columns
vactiveCol=subVactiveCol{is};
% extract report weight data rows
find(vactiveRow)
wUse=w(find(vactiveRow),:)
wRollUpStep1=(wUse'*subAdjUse)'
wRollUpStep=wRollUpStep1+w(find(vactiveCol),:)
w(find(vactiveCol),:)
wRollUp{is}=wRollUpStep
wAccumulate(find(vactiveCol),:)=wAccumulate(find(vactiveCol),:)+wRollUpStep
wAccStep{is}=wAccumulate;
end

```

```
subAdjUse =
  1      1      1
ans =
  1
wUse =
  1      1      0      0      0      0
wRollUpStep1 =
  1      1      0      0      0      0
  1      1      0      0      0      0
  1      1      0      0      0      0
wRollUpStep =
  2      2      0      0      0      0
  1      1      1      0      0      0
  1      1      0      1      0      0
```

```

ans =
  1   1   0   0   0   0
  0   0   1   0   0   0
  0   0   0   1   0   0

wRollUp =
[3x6 double]
wAccumulate =
  0   0   0   0   0   0
  2   2   0   0   0   0
  1   1   1   0   0   0
  0   0   0   0   0   0
  0   0   0   0   0   0
  0   0   0   0   0   0
  1   1   0   1   0   0
  0   0   0   0   0   0
  0   0   0   0   0   0
  0   0   0   0   0   0
  0   0   0   0   0   0
  0   0   0   0   0   0
  0   0   0   0   0   0
  0   0   0   0   0   0
  0   0   0   0   0   0
  0   0   0   0   0   0
  0   0   0   0   0   0
  0   0   0   0   0   0
  0   0   0   0   0   0
  0   0   0   0   0   0
  0   0   0   0   0   0
  0   0   0   0   0   0
  0   0   0   0   0   0

subAdjUse =
  1   1   1   1   0   0
  0   0   0   1   0   0
  1   1   0   0   0   1
  0   0   1   0   1   0

ans =
  1
  2
  3
  7

wUse =
  1   1   0   0   0   0
  1   1   0   0   0   0
  0   0   1   0   0   0
  0   0   0   1   0   0

wRollUpStep1 =
  1   1   1   0   0   0
  1   1   1   0   0   0
  1   1   0   1   0   0
  2   2   0   0   0   0
  0   0   0   1   0   0
  0   0   1   0   0   0

wRollUpStep =
  1   1   1   0   1   0
  1   1   1   0   1   0
  1   1   0   1   1   0
  2   3   0   0   0   0
  0   0   0   2   0   0
  0   0   2   0   0   0

ans =
  0   0   0   0   1   0
  0   0   0   0   1   0
  0   0   0   0   1   0
  0   1   0   0   0   0
  0   0   0   1   0   0
  0   0   1   0   0   0

wRollUp =
[3x6 double]   [6x6 double]
wAccumulate =
  0   0   0   0   0   0
  2   2   0   0   0   0
  1   1   1   0   0   0

```



```
1 1 1 0 1 0
1 1 1 0 1 0
0 0 0 0 0 0
1 1 0 1 0 0
1 1 0 1 1 0
2 3 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 2 0 0
0 0 2 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
subAdjUse =
0 0 0 0 0 1 0
0 0 0 0 0 1 0
1 0 0 0 1 0 0
0 1 0 0 0 0 0
1 0 1 0 0 0 0
0 0 0 0 0 0 1
0 0 0 1 0 0 0
0 0 0 0 0 1 0
0 0 0 0 0 0 1
0 0 0 0 1 0 0
ans =
1
2
3
4
5
7
8
9
14
15
wUse =
1 1 0 0 0 0
1 1 0 0 0 0
0 0 1 0 0 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 1 0 0
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 1 0 0 0
wRollUpStep1 =
0 0 1 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 2 0 0 0
2 3 0 0 0 0
0 0 0 2 0 0
wRollUpStep =
0 0 1 0 2 0
0 0 0 0 2 0
0 0 0 0 2 0
0 0 0 0 2 0
0 0 2 0 0 1
2 3 0 0 0 1
0 0 0 2 0 1
ans =
0 0 0 0 1 0
0 0 0 0 1 0
```

```

0      0      0      0      1      0
0      0      0      0      1      0
0      0      0      0      0      1
0      0      0      0      0      1
0      0      0      0      0      1
wRollUp =
  [3x6 double]    [6x6 double]    [7x6 double]
wAccumulate =
0      0      0      0      0      0
2      2      0      0      0      0
1      1      1      0      0      0
1      1      1      0      1      0
1      1      1      0      1      0
0      0      1      0      2      0
1      1      0      1      0      0
1      1      0      1      1      0
2      3      0      0      0      0
0      0      0      0      2      0
0      0      0      0      2      0
0      0      0      0      0      0
0      0      0      0      2      0
0      0      0      2      0      0
0      0      0      2      0      0
0      0      2      0      0      1
2      3      0      0      0      1
0      0      0      0      0      0
0      0      0      0      0      0
0      0      0      0      0      0
0      0      0      0      0      0
0      0      0      2      0      1
subAdjUse =
0      1      0      0
1      0      1      0
1      0      0      0
0      0      0      1
0      1      0      0
0      0      1      0
0      0      0      1
ans =
4
5
6
8
10
11
13
wUse =
0      0      0      0      1      0
0      0      0      0      1      0
0      0      0      0      1      0
0      0      0      0      1      0
0      0      0      0      1      0
0      0      0      0      1      0
0      0      0      0      1      0
wRollUpStep1 =
0      0      0      0      2      0
0      0      0      0      2      0
0      0      0      0      2      0
0      0      0      0      2      0
wRollUpStep =
0      0      0      0      3      0
0      0      0      0      2      1
0      0      0      0      2      1
0      0      0      0      2      1
ans =
0      0      0      0      1      0
0      0      0      0      0      1
0      0      0      0      0      1
0      0      0      0      0      1
wRollUp =
  Columns 1 through 3

```

```

    [3x6 double]    [6x6 double]    [7x6 double]
Column 4
    [4x6 double]
wAccumulate =
    0    0    0    0    0    0
    2    2    0    0    0    0
    1    1    1    0    0    0
    1    1    1    0    1    0
    1    1    1    0    1    0
    0    0    1    0    2    0
    1    1    0    1    0    0
    1    1    0    1    1    0
    2    3    0    0    0    0
    0    0    0    0    2    0
    0    0    0    0    2    0
    0    0    0    0    3    0
    0    0    0    0    2    0
    0    0    0    2    0    0
    0    0    2    0    0    0
    0    0    2    0    0    1
    2    3    0    0    0    1
    0    0    0    0    2    1
    0    0    0    0    2    1
    0    0    0    0    0    0
    0    0    0    0    2    1
    0    0    0    2    0    1
subAdjUse =
    1
    1
ans =
    6
    12
wUse =
    0    0    0    0    1    0
    0    0    0    0    1    0
wRollUpStep1 =
    0    0    0    0    2    0
wRollUpStep =
    0    0    0    0    2    1
ans =
    0    0    0    0    0    1
wRollUp =
Columns 1 through 3
    [3x6 double]    [6x6 double]    [7x6 double]
Columns 4 through 5
    [4x6 double]    [1x6 double]
wAccumulate =
    0    0    0    0    0    0
    2    2    0    0    0    0
    1    1    1    0    0    0
    1    1    1    0    1    0
    1    1    1    0    1    0
    0    0    1    0    2    0
    1    1    0    1    0    0
    1    1    0    1    1    0
    2    3    0    0    0    0
    0    0    0    0    2    0
    0    0    0    0    2    0
    0    0    0    0    3    0
    0    0    0    0    2    0
    0    0    0    2    0    0
    0    0    2    0    0    0
    0    0    2    0    0    1
    2    3    0    0    0    1
    0    0    0    0    2    1
    0    0    0    0    2    1
    0    0    0    0    2    1
    0    0    0    2    0    1

```

K.9.4 Larger example with string literal values concatenated in accumulation process

Contents

- Read in graph data from yEd .tgf file
- Topological sorting of graph and set up level labels
- Extract sub adjacency matrices and display
- Set up reporting weight and accumulation strings
- Roll up reporting string data

```
%.....
% ReportResultsRollUpTopolStringEx3.m
% Report results roll up Strings Example
%.....
clear all
clc
format compact
```

Read in graph data from yEd .tgf file

graph of reporting structure - read ex data file

```
format compact
tgfFile='PEPEXTsequenceT.tgf'
[vertexLabels,edgeLabels,TreeAdj] = TgfRead(tgfFile)
%tgfFilePre='TSCEX3'
% Output graph for yEd display - not used
% Tree adjacency matrix
%isHomog=1
%fileTC=horzcat(tgfFilePre,'TreeAdjConne','tgf')
%tgfWrite(fileTC,TreeAdj,isHomog,vertexLabels,{});
% scratch last row & column of adjacency matrix
% contains header of graph - not necessary - no edge links vertex
% TreeAdj=TreeAdj(1:size(TreeAdj,1)-1,1:(size(TreeAdj,2)-1))
% Topological sorting of tree vertices
```

```
tgfFile =
PEPEXTsequenceT.tgf
File PEPEXTsequenceT.tgf opened
File PEPEXTsequenceT.tgf closed
vertexLabels =
  Columns 1 through 4
    [1x38 char]    [1x38 char]    [1x38 char]    [1x38 char]
  Columns 5 through 8
    [1x38 char]    [1x38 char]    [1x38 char]    [1x38 char]
  Columns 9 through 12
    [1x38 char]    [1x38 char]    [1x38 char]    [1x38 char]
  Columns 13 through 16
    [1x38 char]    [1x38 char]    [1x38 char]    [1x38 char]
  Columns 17 through 20
    [1x38 char]    [1x38 char]    [1x38 char]    [1x38 char]
  Columns 21 through 23
    [1x38 char]    [1x38 char]    [1x39 char]
edgeLabels =
  Columns 1 through 8
    , , , , , , , ,
  Columns 9 through 16
    , , , , , , , ,
  Columns 17 through 24
    , , , , , , , ,
  Columns 25 through 32
    , , , , , , , ,
  Columns 33 through 35
    , , , , ,
```

```
TreeAdj =
Columns 1 through 10
0 1 1 1 1 0 1 1 1 0
0 0 0 0 0 0 0 0 1 0
0 0 0 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
Columns 11 through 20
0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 1 0 0 0
0 0 0 0 1 1 0 0 0 0
0 0 0 0 0 0 0 1 0 0
1 1 0 0 0 0 0 0 1 0
0 1 0 0 0 0 0 0 0 1
0 0 0 1 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
Columns 21 through 23
0 0 0
0 0 0
0 0 0
0 0 0
0 0 0
0 0 0
0 1 0
1 0 0
0 0 0
0 0 0
0 0 0
0 0 0
1 0 0
0 1 0
0 0 0
0 0 0
0 0 0
0 0 0
0 0 0
0 0 0
0 0 0
```

Topological sorting of graph and set up level labels

$$\begin{array}{ccc} 0 & 0 & 0 \end{array}$$


```
% 'Client          p01', ...
% 'Architect       p02', ...
% 'Structural Engineer p03', ...
% 'Electrical Engineer p04', ...
% 'Draftsman        p05', ...
% 'Checking Engineer p06',
```

```
nRow =
    23
nCol =
     6
subAdj =
    Columns 1 through 3
    [1x3 double]    [4x6 double]    [10x7 double]
    Columns 4 through 5
    [7x4 double]    [2x1 double]
subVactiveCol =
    Columns 1 through 3
    [1x7 double]    [1x15 double]    [1x22 double]
    Columns 4 through 5
    [1x21 double]    [1x20 double]
subVactiveRow =
    Columns 1 through 3
    [23x1 double]    [23x1 double]    [23x1 double]
    Columns 4 through 5
    [23x1 double]    [23x1 double]
subAdj{1} =
     1     1     1
subAdj{2} =
     1     1     1     1     0     0
     0     0     0     1     0     0
     1     1     0     0     0     1
     0     0     1     0     1     0
subAdj{3} =
     0     0     0     0     0     1     0
     0     0     0     0     0     1     0
     1     0     0     0     1     0     0
     0     1     0     0     0     0     0
     1     0     1     0     0     0     0
     0     0     0     0     0     0     1
     0     0     0     1     0     0     0
     0     0     0     0     0     1     0
     0     0     0     0     0     0     1
     0     0     0     0     1     0     0
subAdj{4} =
     0     1     0     0
     1     0     1     0
     1     0     0     0
     0     0     0     1
     0     1     0     0
     0     0     1     0
     0     0     0     1
subAdj{5} =
     1
     1
subVactiveCol{1} =
     0     1     1     0     0     0     1
subVactiveCol{2} =
    Columns 1 through 10
     0     0     0     1     1     0     0     1     1     0
    Columns 11 through 15
     0     0     0     1     1
subVactiveCol{3} =
    Columns 1 through 10
     0     0     0     0     0     1     0     0     0     1
    Columns 11 through 20
     1     0     1     0     0     1     1     0     0     0
    Columns 21 through 22
```


[illegible]

```

0
0
0
0
1
1
0
0
0
0
0
0
0
0
0
0
subVactiveRow{4} =
0
0
0
1
1
1
1
0
1
0
1
0
1
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
subVactiveRow{5} =
0
0
0
0
0
0
1
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0

```

Set up reporting weight and accumulation strings

```

nW=6
wString={
' t01p1' , ' t01p2' , ' t01p3' , ' t01p4' , ' t01p5' , ' t01p6' ;

```

```

' t02p1' ' t02p2' ' t02p3' ' t02p4' ' t02p5' ' t02p6' ;
' t03p1' ' t03p2' ' t03p3' ' t03p4' ' t03p5' ' t03p6' ;
' t04p1' ' t04p2' ' t04p3' ' t04p4' ' t04p5' ' t04p6' ;
' t05p1' ' t05p2' ' t05p3' ' t05p4' ' t05p5' ' t05p6' ;
' t06p1' ' t06p2' ' t06p3' ' t06p4' ' t06p5' ' t06p6' ;
' t07p1' ' t07p2' ' t07p3' ' t07p4' ' t07p5' ' t07p6' ;
' t08p1' ' t08p2' ' t08p3' ' t08p4' ' t08p5' ' t08p6' ;
' t09p1' ' t09p2' ' t09p3' ' t09p4' ' t09p5' ' t09p6' ;
' t10p1' ' t10p2' ' t10p3' ' t10p4' ' t10p5' ' t10p6' ;
' t11p1' ' t11p2' ' t11p3' ' t11p4' ' t11p5' ' t11p6' ;
' t12p1' ' t12p2' ' t12p3' ' t12p4' ' t12p5' ' t12p6' ;
' t13p1' ' t13p2' ' t13p3' ' t13p4' ' t13p5' ' t13p6' ;
' t14p1' ' t14p2' ' t14p3' ' t14p4' ' t14p5' ' t14p6' ;
' t15p1' ' t15p2' ' t15p3' ' t15p4' ' t15p5' ' t15p6' ;
' t16p1' ' t16p2' ' t16p3' ' t16p4' ' t16p5' ' t16p6' ;
' t17p1' ' t17p2' ' t17p3' ' t17p4' ' t17p5' ' t17p6' ;
' t18p1' ' t18p2' ' t18p3' ' t18p4' ' t18p5' ' t18p6' ;
' t19p1' ' t19p2' ' t19p3' ' t19p4' ' t19p5' ' t19p6' ;
' t20p1' ' t20p2' ' t20p3' ' t20p4' ' t20p5' ' t20p6' ;
' t21p1' ' t21p2' ' t21p3' ' t21p4' ' t21p5' ' t21p6' ;
' t22p1' ' t22p2' ' t22p3' ' t22p4' ' t22p5' ' t22p6' ;
}
% set accumulation array to blank strings
for ir=1:(size(TreeAdj,1)-1)
    for ic=1:nW
        wAccumulate{ir,ic}='';
    end
end
% number of submatrices - only use to limit output
nsub=size(subAdj,2)
nsubUse=nsub
%nsubUse=2

```

```

nW =
    6
wString =
    Columns 1 through 5
    ' t01p1' ' t01p2' ' t01p3' ' t01p4' ' t01p5'
    ' t02p1' ' t02p2' ' t02p3' ' t02p4' ' t02p5'
    ' t03p1' ' t03p2' ' t03p3' ' t03p4' ' t03p5'
    ' t04p1' ' t04p2' ' t04p3' ' t04p4' ' t04p5'
    ' t05p1' ' t05p2' ' t05p3' ' t05p4' ' t05p5'
    ' t06p1' ' t06p2' ' t06p3' ' t06p4' ' t06p5'
    ' t07p1' ' t07p2' ' t07p3' ' t07p4' ' t07p5'
    ' t08p1' ' t08p2' ' t08p3' ' t08p4' ' t08p5'
    ' t09p1' ' t09p2' ' t09p3' ' t09p4' ' t09p5'
    ' t10p1' ' t10p2' ' t10p3' ' t10p4' ' t10p5'
    ' t11p1' ' t11p2' ' t11p3' ' t11p4' ' t11p5'
    ' t12p1' ' t12p2' ' t12p3' ' t12p4' ' t12p5'
    ' t13p1' ' t13p2' ' t13p3' ' t13p4' ' t13p5'
    ' t14p1' ' t14p2' ' t14p3' ' t14p4' ' t14p5'
    ' t15p1' ' t15p2' ' t15p3' ' t15p4' ' t15p5'
    ' t16p1' ' t16p2' ' t16p3' ' t16p4' ' t16p5'
    ' t17p1' ' t17p2' ' t17p3' ' t17p4' ' t17p5'
    ' t18p1' ' t18p2' ' t18p3' ' t18p4' ' t18p5'
    ' t19p1' ' t19p2' ' t19p3' ' t19p4' ' t19p5'
    ' t20p1' ' t20p2' ' t20p3' ' t20p4' ' t20p5'
    ' t21p1' ' t21p2' ' t21p3' ' t21p4' ' t21p5'
    ' t22p1' ' t22p2' ' t22p3' ' t22p4' ' t22p5'
    Column 6
    ' t01p6'
    ' t02p6'
    ' t03p6'
    ' t04p6'
    ' t05p6'
    ' t06p6'
    ' t07p6'
    ' t08p6'

```

```

    ' t09p6'
    ' t10p6'
    ' t11p6'
    ' t12p6'
    ' t13p6'
    ' t14p6'
    ' t15p6'
    ' t16p6'
    ' t17p6'
    ' t18p6'
    ' t19p6'
    ' t20p6'
    ' t21p6'
    ' t22p6'
nsub =
    5
nsubUse =
    5

```

Roll up reporting string data

```

for is=1:nsubUse
% Roll Up from Low to High
% adjacency matrix
subAdjUse=subAdj{is}
% active rows
vactiveRow=subVactiveRow{is};
% active columns
vactiveCol=subVactiveCol{is};
% extract report weight data rows
fndVactiveRow=find(vactiveRow)
wUse=wString(find(vactiveRow),:)
% wRollUpStep1=(wUse'*subAdjUse)'
wRollUpStep1=stringMultBool(wUse',subAdjUse)'
wRollUpIncrement=wString(find(vactiveCol),:)
wRollUpStep=strcat(wRollUpStep1,wRollUpIncrement)
wRollUp{is}=wRollUpStep
wAccumulate(find(vactiveCol),:)= ...
    strcat(wAccumulate(find(vactiveCol),:),wRollUpStep)
wAccStep{is}=wAccumulate;
end

```

```

subAdjUse =
    1    1    1
fndVactiveRow =
    1
wUse =
    Columns 1 through 5
    ' t01p1'    ' t01p2'    ' t01p3'    ' t01p4'    ' t01p5'
    Column 6
    ' t01p6'
wRollUpStep1 =
    Columns 1 through 5
    ' t01p1'    ' t01p2'    ' t01p3'    ' t01p4'    ' t01p5'
    ' t01p1'    ' t01p2'    ' t01p3'    ' t01p4'    ' t01p5'
    ' t01p1'    ' t01p2'    ' t01p3'    ' t01p4'    ' t01p5'
    Column 6
    ' t01p6'
    ' t01p6'
    ' t01p6'
wRollUpIncrement =
    Columns 1 through 5
    ' t02p1'    ' t02p2'    ' t02p3'    ' t02p4'    ' t02p5'
    ' t03p1'    ' t03p2'    ' t03p3'    ' t03p4'    ' t03p5'
    ' t07p1'    ' t07p2'    ' t07p3'    ' t07p4'    ' t07p5'

```

```

Column 6
' t02p6'
' t03p6'
' t07p6'
wRollUpStep =
Columns 1 through 3
' t01p1 t02p1' ' t01p2 t02p2' ' t01p3 t02p3'
' t01p1 t03p1' ' t01p2 t03p2' ' t01p3 t03p3'
' t01p1 t07p1' ' t01p2 t07p2' ' t01p3 t07p3'
Columns 4 through 6
' t01p4 t02p4' ' t01p5 t02p5' ' t01p6 t02p6'
' t01p4 t03p4' ' t01p5 t03p5' ' t01p6 t03p6'
' t01p4 t07p4' ' t01p5 t07p5' ' t01p6 t07p6'
wRollUp =
{3x6 cell}
wAccumulate =
Columns 1 through 3
'' '' ''
' t01p1 t02p1' ' t01p2 t02p2' ' t01p3 t02p3'
' t01p1 t03p1' ' t01p2 t03p2' ' t01p3 t03p3'
'' '' ''
'' '' ''
'' '' ''
' t01p1 t07p1' ' t01p2 t07p2' ' t01p3 t07p3'
'' '' ''
'' '' ''
'' '' ''
'' '' ''
'' '' ''
'' '' ''
'' '' ''
'' '' ''
'' '' ''
'' '' ''
'' '' ''
'' '' ''
'' '' ''
'' '' ''
'' '' ''
'' '' ''
Columns 4 through 6
'' '' ''
' t01p4 t02p4' ' t01p5 t02p5' ' t01p6 t02p6'
' t01p4 t03p4' ' t01p5 t03p5' ' t01p6 t03p6'
'' '' ''
'' '' ''
'' '' ''
' t01p4 t07p4' ' t01p5 t07p5' ' t01p6 t07p6'
'' '' ''
'' '' ''
'' '' ''
'' '' ''
'' '' ''
'' '' ''
'' '' ''
'' '' ''
'' '' ''
'' '' ''
'' '' ''
'' '' ''
'' '' ''
'' '' ''
'' '' ''
'' '' ''
'' '' ''
'' '' ''
subAdjUse =
1 1 1 1 0 0
0 0 0 1 0 0
1 1 0 0 0 1
0 0 1 0 1 0
fndVactiveRow =
1
2
3

```

```

7
wUse =
  Columns 1 through 5
    ' t01p1'    ' t01p2'    ' t01p3'    ' t01p4'    ' t01p5'
    ' t02p1'    ' t02p2'    ' t02p3'    ' t02p4'    ' t02p5'
    ' t03p1'    ' t03p2'    ' t03p3'    ' t03p4'    ' t03p5'
    ' t07p1'    ' t07p2'    ' t07p3'    ' t07p4'    ' t07p5'
  Column 6
    ' t01p6'
    ' t02p6'
    ' t03p6'
    ' t07p6'
wRollUpStep1 =
  Columns 1 through 3
    ' t01p1 t03p1'    ' t01p2 t03p2'    ' t01p3 t03p3'
    ' t01p1 t03p1'    ' t01p2 t03p2'    ' t01p3 t03p3'
    ' t01p1 t07p1'    ' t01p2 t07p2'    ' t01p3 t07p3'
    ' t01p1 t02p1'    ' t01p2 t02p2'    ' t01p3 t02p3'
    ' t07p1'          ' t07p2'          ' t07p3'
    ' t03p1'          ' t03p2'          ' t03p3'
  Columns 4 through 6
    ' t01p4 t03p4'    ' t01p5 t03p5'    ' t01p6 t03p6'
    ' t01p4 t03p4'    ' t01p5 t03p5'    ' t01p6 t03p6'
    ' t01p4 t07p4'    ' t01p5 t07p5'    ' t01p6 t07p6'
    ' t01p4 t02p4'    ' t01p5 t02p5'    ' t01p6 t02p6'
    ' t07p4'          ' t07p5'          ' t07p6'
    ' t03p4'          ' t03p5'          ' t03p6'
wRollUpIncrement =
  Columns 1 through 5
    ' t04p1'    ' t04p2'    ' t04p3'    ' t04p4'    ' t04p5'
    ' t05p1'    ' t05p2'    ' t05p3'    ' t05p4'    ' t05p5'
    ' t08p1'    ' t08p2'    ' t08p3'    ' t08p4'    ' t08p5'
    ' t09p1'    ' t09p2'    ' t09p3'    ' t09p4'    ' t09p5'
    ' t14p1'    ' t14p2'    ' t14p3'    ' t14p4'    ' t14p5'
    ' t15p1'    ' t15p2'    ' t15p3'    ' t15p4'    ' t15p5'
  Column 6
    ' t04p6'
    ' t05p6'
    ' t08p6'
    ' t09p6'
    ' t14p6'
    ' t15p6'
wRollUpStep =
  Columns 1 through 2
    ' t01p1 t03p1 t04p1'    ' t01p2 t03p2 t04p2'
    ' t01p1 t03p1 t05p1'    ' t01p2 t03p2 t05p2'
    ' t01p1 t07p1 t08p1'    ' t01p2 t07p2 t08p2'
    ' t01p1 t02p1 t09p1'    ' t01p2 t02p2 t09p2'
    ' t07p1 t14p1'          ' t07p2 t14p2'
    ' t03p1 t15p1'          ' t03p2 t15p2'
  Columns 3 through 4
    ' t01p3 t03p3 t04p3'    ' t01p4 t03p4 t04p4'
    ' t01p3 t03p3 t05p3'    ' t01p4 t03p4 t05p4'
    ' t01p3 t07p3 t08p3'    ' t01p4 t07p4 t08p4'
    ' t01p3 t02p3 t09p3'    ' t01p4 t02p4 t09p4'
    ' t07p3 t14p3'          ' t07p4 t14p4'
    ' t03p3 t15p3'          ' t03p4 t15p4'
  Columns 5 through 6
    ' t01p5 t03p5 t04p5'    ' t01p6 t03p6 t04p6'
    ' t01p5 t03p5 t05p5'    ' t01p6 t03p6 t05p6'
    ' t01p5 t07p5 t08p5'    ' t01p6 t07p6 t08p6'
    ' t01p5 t02p5 t09p5'    ' t01p6 t02p6 t09p6'
    ' t07p5 t14p5'          ' t07p6 t14p6'
    ' t03p5 t15p5'          ' t03p6 t15p6'
wRollUp =
  {3x6 cell}    {6x6 cell}
wAccumulate =
  Columns 1 through 2
    , ,
    ' t01p1 t02p1'    ' t01p2 t02p2'
    ' t01p1 t03p1'    ' t01p2 t03p2'

```

```

' t01p1 t03p1 t04p1'      ' t01p2 t03p2 t04p2'
' t01p1 t03p1 t05p1'      ' t01p2 t03p2 t05p2'
      ,,                      ,,

' t01p1 t07p1'            ' t01p2 t07p2'
' t01p1 t07p1 t08p1'      ' t01p2 t07p2 t08p2'
' t01p1 t02p1 t09p1'      ' t01p2 t02p2 t09p2'
      ,,                      ,,
      ,,                      ,,
      ,,                      ,,
      ,,                      ,,

' t07p1 t14p1'            ' t07p2 t14p2'
' t03p1 t15p1'            ' t03p2 t15p2'
      ,,                      ,,
      ,,                      ,,
      ,,                      ,,
      ,,                      ,,
      ,,                      ,,
      ,,                      ,,
      ,,                      ,,

Columns 3 through 4
      ,,                      ,,

' t01p3 t02p3'            ' t01p4 t02p4'
' t01p3 t03p3'            ' t01p4 t03p4'
' t01p3 t03p3 t04p3'      ' t01p4 t03p4 t04p4'
' t01p3 t03p3 t05p3'      ' t01p4 t03p4 t05p4'
      ,,                      ,,
      ,,                      ,,
      ,,                      ,,
      ,,                      ,,
      ,,                      ,,
      ,,                      ,,
      ,,                      ,,

' t07p3 t14p3'            ' t07p4 t14p4'
' t03p3 t15p3'            ' t03p4 t15p4'
      ,,                      ,,
      ,,                      ,,
      ,,                      ,,
      ,,                      ,,
      ,,                      ,,
      ,,                      ,,
      ,,                      ,,

Columns 5 through 6
      ,,                      ,,

' t01p5 t02p5'            ' t01p6 t02p6'
' t01p5 t03p5'            ' t01p6 t03p6'
' t01p5 t03p5 t04p5'      ' t01p6 t03p6 t04p6'
' t01p5 t03p5 t05p5'      ' t01p6 t03p6 t05p6'
      ,,                      ,,
      ,,                      ,,
      ,,                      ,,
      ,,                      ,,
      ,,                      ,,
      ,,                      ,,
      ,,                      ,,

' t07p5 t14p5'            ' t07p6 t14p6'
' t03p5 t15p5'            ' t03p6 t15p6'
      ,,                      ,,
      ,,                      ,,
      ,,                      ,,
      ,,                      ,,
      ,,                      ,,
      ,,                      ,,
      ,,                      ,,

subAdjUse =
0      0      0      0      0      1      0
0      0      0      0      0      1      0
1      0      0      0      1      0      0
0      1      0      0      0      0      0
1      0      1      0      0      0      0

```

```

0      0      0      0      0      0      1
0      0      0      1      0      0      0
0      0      0      0      0      1      0
0      0      0      0      0      0      1
0      0      0      0      1      0      0
fndVactiveRow =
1
2
3
4
5
7
8
9
14
15
wUse =
Columns 1 through 5
' t01p1'      ' t01p2'      ' t01p3'      ' t01p4'      ' t01p5'
' t02p1'      ' t02p2'      ' t02p3'      ' t02p4'      ' t02p5'
' t03p1'      ' t03p2'      ' t03p3'      ' t03p4'      ' t03p5'
' t04p1'      ' t04p2'      ' t04p3'      ' t04p4'      ' t04p5'
' t05p1'      ' t05p2'      ' t05p3'      ' t05p4'      ' t05p5'
' t07p1'      ' t07p2'      ' t07p3'      ' t07p4'      ' t07p5'
' t08p1'      ' t08p2'      ' t08p3'      ' t08p4'      ' t08p5'
' t09p1'      ' t09p2'      ' t09p3'      ' t09p4'      ' t09p5'
' t14p1'      ' t14p2'      ' t14p3'      ' t14p4'      ' t14p5'
' t15p1'      ' t15p2'      ' t15p3'      ' t15p4'      ' t15p5'
Column 6
' t01p6'
' t02p6'
' t03p6'
' t04p6'
' t05p6'
' t07p6'
' t08p6'
' t09p6'
' t14p6'
' t15p6'
wRollUpStep1 =
Columns 1 through 2
' t03p1 t05p1'      ' t03p2 t05p2'
' t04p1'      ' t04p2'
' t05p1'      ' t05p2'
' t08p1'      ' t08p2'
' t03p1 t15p1'      ' t03p2 t15p2'
' t01p1 t02p1 t09p1'      ' t01p2 t02p2 t09p2'
' t07p1 t14p1'      ' t07p2 t14p2'
Columns 3 through 4
' t03p3 t05p3'      ' t03p4 t05p4'
' t04p3'      ' t04p4'
' t05p3'      ' t05p4'
' t08p3'      ' t08p4'
' t03p3 t15p3'      ' t03p4 t15p4'
' t01p3 t02p3 t09p3'      ' t01p4 t02p4 t09p4'
' t07p3 t14p3'      ' t07p4 t14p4'
Columns 5 through 6
' t03p5 t05p5'      ' t03p6 t05p6'
' t04p5'      ' t04p6'
' t05p5'      ' t05p6'
' t08p5'      ' t08p6'
' t03p5 t15p5'      ' t03p6 t15p6'
' t01p5 t02p5 t09p5'      ' t01p6 t02p6 t09p6'
' t07p5 t14p5'      ' t07p6 t14p6'
wRollUpIncrement =
Columns 1 through 5
' t06p1'      ' t06p2'      ' t06p3'      ' t06p4'      ' t06p5'
' t10p1'      ' t10p2'      ' t10p3'      ' t10p4'      ' t10p5'
' t11p1'      ' t11p2'      ' t11p3'      ' t11p4'      ' t11p5'
' t13p1'      ' t13p2'      ' t13p3'      ' t13p4'      ' t13p5'
' t16p1'      ' t16p2'      ' t16p3'      ' t16p4'      ' t16p5'

```



```

    ' t17p1'    ' t17p2'    ' t17p3'    ' t17p4'    ' t17p5'
    ' t22p1'    ' t22p2'    ' t22p3'    ' t22p4'    ' t22p5'
Column 6
    ' t06p6'
    ' t10p6'
    ' t11p6'
    ' t13p6'
    ' t16p6'
    ' t17p6'
    ' t22p6'
wRollUpStep =
Columns 1 through 2
    ' t03p1 t05p1 t06p1'    ' t03p2 t05p2 t06p2'
    ' t04p1 t10p1'          ' t04p2 t10p2'
    ' t05p1 t11p1'          ' t05p2 t11p2'
    ' t08p1 t13p1'          ' t08p2 t13p2'
    ' t03p1 t15p1 t16p1'    ' t03p2 t15p2 t16p2'
        [1x24 char]          [1x24 char]
    ' t07p1 t14p1 t22p1'    ' t07p2 t14p2 t22p2'
Columns 3 through 4
    ' t03p3 t05p3 t06p3'    ' t03p4 t05p4 t06p4'
    ' t04p3 t10p3'          ' t04p4 t10p4'
    ' t05p3 t11p3'          ' t05p4 t11p4'
    ' t08p3 t13p3'          ' t08p4 t13p4'
    ' t03p3 t15p3 t16p3'    ' t03p4 t15p4 t16p4'
        [1x24 char]          [1x24 char]
    ' t07p3 t14p3 t22p3'    ' t07p4 t14p4 t22p4'
Columns 5 through 6
    ' t03p5 t05p5 t06p5'    ' t03p6 t05p6 t06p6'
    ' t04p5 t10p5'          ' t04p6 t10p6'
    ' t05p5 t11p5'          ' t05p6 t11p6'
    ' t08p5 t13p5'          ' t08p6 t13p6'
    ' t03p5 t15p5 t16p5'    ' t03p6 t15p6 t16p6'
        [1x24 char]          [1x24 char]
    ' t07p5 t14p5 t22p5'    ' t07p6 t14p6 t22p6'
wRollUp =
    {3x6 cell}    {6x6 cell}    {7x6 cell}
wAccumulate =
Columns 1 through 2
    ' t01p1 t02p1'    ' t01p2 t02p2'
    ' t01p1 t03p1'    ' t01p2 t03p2'
    ' t01p1 t03p1 t04p1'    ' t01p2 t03p2 t04p2'
    ' t01p1 t03p1 t05p1'    ' t01p2 t03p2 t05p2'
    ' t03p1 t05p1 t06p1'    ' t03p2 t05p2 t06p2'
    ' t01p1 t07p1'          ' t01p2 t07p2'
    ' t01p1 t07p1 t08p1'    ' t01p2 t07p2 t08p2'
    ' t01p1 t02p1 t09p1'    ' t01p2 t02p2 t09p2'
    ' t04p1 t10p1'          ' t04p2 t10p2'
    ' t05p1 t11p1'          ' t05p2 t11p2'
    ' t08p1 t13p1'          ' t08p2 t13p2'
    ' t07p1 t14p1'          ' t07p2 t14p2'
    ' t03p1 t15p1'          ' t03p2 t15p2'
    ' t03p1 t15p1 t16p1'    ' t03p2 t15p2 t16p2'
        [1x24 char]          [1x24 char]
    ' t07p1 t14p1 t22p1'    ' t07p2 t14p2 t22p2'
Columns 3 through 4
    ' t01p3 t02p3'    ' t01p4 t02p4'
    ' t01p3 t03p3'    ' t01p4 t03p4'
    ' t01p3 t03p3 t04p3'    ' t01p4 t03p4 t04p4'
    ' t01p3 t03p3 t05p3'    ' t01p4 t03p4 t05p4'
    ' t03p3 t05p3 t06p3'    ' t03p4 t05p4 t06p4'
    ' t01p3 t07p3'          ' t01p4 t07p4'
    ' t01p3 t07p3 t08p3'    ' t01p4 t07p4 t08p4'
    ' t01p3 t02p3 t09p3'    ' t01p4 t02p4 t09p4'

```

```

' t04p3 t10p3'      ' t04p4 t10p4'
' t05p3 t11p3'      ' t05p4 t11p4'
      ,,              ,,

' t08p3 t13p3'      ' t08p4 t13p4'
' t07p3 t14p3'      ' t07p4 t14p4'
' t03p3 t15p3'      ' t03p4 t15p4'
' t03p3 t15p3 t16p3' ' t03p4 t15p4 t16p4'
      [1x24 char]      [1x24 char]
      ,,              ,,
      ,,              ,,
      ,,              ,,
      ,,              ,,

' t07p3 t14p3 t22p3' ' t07p4 t14p4 t22p4'
Columns 5 through 6      ,,
      ,,              ,,

' t01p5 t02p5'      ' t01p6 t02p6'
' t01p5 t03p5'      ' t01p6 t03p6'
' t01p5 t03p5 t04p5' ' t01p6 t03p6 t04p6'
' t01p5 t03p5 t05p5' ' t01p6 t03p6 t05p6'
' t03p5 t05p5 t06p5' ' t03p6 t05p6 t06p6'
' t01p5 t07p5'      ' t01p6 t07p6'
' t01p5 t07p5 t08p5' ' t01p6 t07p6 t08p6'
' t01p5 t02p5 t09p5' ' t01p6 t02p6 t09p6'
' t04p5 t10p5'      ' t04p6 t10p6'
' t05p5 t11p5'      ' t05p6 t11p6'
      ,,              ,,

' t08p5 t13p5'      ' t08p6 t13p6'
' t07p5 t14p5'      ' t07p6 t14p6'
' t03p5 t15p5'      ' t03p6 t15p6'
' t03p5 t15p5 t16p5' ' t03p6 t15p6 t16p6'
      [1x24 char]      [1x24 char]
      ,,              ,,
      ,,              ,,
      ,,              ,,
      ,,              ,,

' t07p5 t14p5 t22p5' ' t07p6 t14p6 t22p6'
subAdjUse =
0      1      0      0
1      0      1      0
1      0      0      0
0      0      0      1
0      1      0      0
0      0      1      0
0      0      0      1
fndVactiveRow =
4
5
6
8
10
11
13
wUse =
Columns 1 through 5
' t04p1'      ' t04p2'      ' t04p3'      ' t04p4'      ' t04p5'
' t05p1'      ' t05p2'      ' t05p3'      ' t05p4'      ' t05p5'
' t06p1'      ' t06p2'      ' t06p3'      ' t06p4'      ' t06p5'
' t08p1'      ' t08p2'      ' t08p3'      ' t08p4'      ' t08p5'
' t10p1'      ' t10p2'      ' t10p3'      ' t10p4'      ' t10p5'
' t11p1'      ' t11p2'      ' t11p3'      ' t11p4'      ' t11p5'
' t13p1'      ' t13p2'      ' t13p3'      ' t13p4'      ' t13p5'
Column 6
' t04p6'
' t05p6'
' t06p6'
' t08p6'
' t10p6'
' t11p6'
' t13p6'
wRollUpStep1 =
Columns 1 through 3

```

```

    ' t05p1 t06p1'    ' t05p2 t06p2'    ' t05p3 t06p3'
    ' t04p1 t10p1'    ' t04p2 t10p2'    ' t04p3 t10p3'
    ' t05p1 t11p1'    ' t05p2 t11p2'    ' t05p3 t11p3'
    ' t08p1 t13p1'    ' t08p2 t13p2'    ' t08p3 t13p3'
Columns 4 through 6
    ' t05p4 t06p4'    ' t05p5 t06p5'    ' t05p6 t06p6'
    ' t04p4 t10p4'    ' t04p5 t10p5'    ' t04p6 t10p6'
    ' t05p4 t11p4'    ' t05p5 t11p5'    ' t05p6 t11p6'
    ' t08p4 t13p4'    ' t08p5 t13p5'    ' t08p6 t13p6'
wRollUpIncrement =
Columns 1 through 5
    ' t12p1'    ' t12p2'    ' t12p3'    ' t12p4'    ' t12p5'
    ' t18p1'    ' t18p2'    ' t18p3'    ' t18p4'    ' t18p5'
    ' t19p1'    ' t19p2'    ' t19p3'    ' t19p4'    ' t19p5'
    ' t21p1'    ' t21p2'    ' t21p3'    ' t21p4'    ' t21p5'
Column 6
    ' t12p6'
    ' t18p6'
    ' t19p6'
    ' t21p6'
wRollUpStep =
Columns 1 through 2
    ' t05p1 t06p1 t12p1'    ' t05p2 t06p2 t12p2'
    ' t04p1 t10p1 t18p1'    ' t04p2 t10p2 t18p2'
    ' t05p1 t11p1 t19p1'    ' t05p2 t11p2 t19p2'
    ' t08p1 t13p1 t21p1'    ' t08p2 t13p2 t21p2'
Columns 3 through 4
    ' t05p3 t06p3 t12p3'    ' t05p4 t06p4 t12p4'
    ' t04p3 t10p3 t18p3'    ' t04p4 t10p4 t18p4'
    ' t05p3 t11p3 t19p3'    ' t05p4 t11p4 t19p4'
    ' t08p3 t13p3 t21p3'    ' t08p4 t13p4 t21p4'
Columns 5 through 6
    ' t05p5 t06p5 t12p5'    ' t05p6 t06p6 t12p6'
    ' t04p5 t10p5 t18p5'    ' t04p6 t10p6 t18p6'
    ' t05p5 t11p5 t19p5'    ' t05p6 t11p6 t19p6'
    ' t08p5 t13p5 t21p5'    ' t08p6 t13p6 t21p6'
wRollUp =
    {3x6 cell}    {6x6 cell}    {7x6 cell}    {4x6 cell}
wAccumulate =
Columns 1 through 2
    , ,
    ' t01p1 t02p1'    ' t01p2 t02p2'
    ' t01p1 t03p1'    ' t01p2 t03p2'
    ' t01p1 t03p1 t04p1'    ' t01p2 t03p2 t04p2'
    ' t01p1 t03p1 t05p1'    ' t01p2 t03p2 t05p2'
    ' t03p1 t05p1 t06p1'    ' t03p2 t05p2 t06p2'
    ' t01p1 t07p1'    ' t01p2 t07p2'
    ' t01p1 t07p1 t08p1'    ' t01p2 t07p2 t08p2'
    ' t01p1 t02p1 t09p1'    ' t01p2 t02p2 t09p2'
    ' t04p1 t10p1'    ' t04p2 t10p2'
    ' t05p1 t11p1'    ' t05p2 t11p2'
    ' t05p1 t06p1 t12p1'    ' t05p2 t06p2 t12p2'
    ' t08p1 t13p1'    ' t08p2 t13p2'
    ' t07p1 t14p1'    ' t07p2 t14p2'
    ' t03p1 t15p1'    ' t03p2 t15p2'
    ' t03p1 t15p1 t16p1'    ' t03p2 t15p2 t16p2'
    [1x24 char]    [1x24 char]
    ' t04p1 t10p1 t18p1'    ' t04p2 t10p2 t18p2'
    ' t05p1 t11p1 t19p1'    ' t05p2 t11p2 t19p2'
    , ,
    ' t08p1 t13p1 t21p1'    ' t08p2 t13p2 t21p2'
    ' t07p1 t14p1 t22p1'    ' t07p2 t14p2 t22p2'
Columns 3 through 4
    , ,
    ' t01p3 t02p3'    ' t01p4 t02p4'
    ' t01p3 t03p3'    ' t01p4 t03p4'
    ' t01p3 t03p3 t04p3'    ' t01p4 t03p4 t04p4'
    ' t01p3 t03p3 t05p3'    ' t01p4 t03p4 t05p4'
    ' t03p3 t05p3 t06p3'    ' t03p4 t05p4 t06p4'
    ' t01p3 t07p3'    ' t01p4 t07p4'
    ' t01p3 t07p3 t08p3'    ' t01p4 t07p4 t08p4'

```

```

    ' t01p3 t02p3 t09p3',    ' t01p4 t02p4 t09p4',
    ' t04p3 t10p3',         ' t04p4 t10p4',
    ' t05p3 t11p3',         ' t05p4 t11p4',
    ' t05p3 t06p3 t12p3',   ' t05p4 t06p4 t12p4',
    ' t08p3 t13p3',         ' t08p4 t13p4',
    ' t07p3 t14p3',         ' t07p4 t14p4',
    ' t03p3 t15p3',         ' t03p4 t15p4',
    ' t03p3 t15p3 t16p3',   ' t03p4 t15p4 t16p4',
    [1x24 char]             [1x24 char]
    ' t04p3 t10p3 t18p3',   ' t04p4 t10p4 t18p4',
    ' t05p3 t11p3 t19p3',   ' t05p4 t11p4 t19p4',
    ',,'                    ',,'
    ' t08p3 t13p3 t21p3',   ' t08p4 t13p4 t21p4',
    ' t07p3 t14p3 t22p3',   ' t07p4 t14p4 t22p4',
Columns 5 through 6      ',,'
    ',,'                    ',,'
    ' t01p5 t02p5',         ' t01p6 t02p6',
    ' t01p5 t03p5',         ' t01p6 t03p6',
    ' t01p5 t03p5 t04p5',   ' t01p6 t03p6 t04p6',
    ' t01p5 t03p5 t05p5',   ' t01p6 t03p6 t05p6',
    ' t03p5 t05p5 t06p5',   ' t03p6 t05p6 t06p6',
    ' t01p5 t07p5',         ' t01p6 t07p6',
    ' t01p5 t07p5 t08p5',   ' t01p6 t07p6 t08p6',
    ' t01p5 t02p5 t09p5',   ' t01p6 t02p6 t09p6',
    ' t04p5 t10p5',         ' t04p6 t10p6',
    ' t05p5 t11p5',         ' t05p6 t11p6',
    ' t05p5 t06p5 t12p5',   ' t05p6 t06p6 t12p6',
    ' t08p5 t13p5',         ' t08p6 t13p6',
    ' t07p5 t14p5',         ' t07p6 t14p6',
    ' t03p5 t15p5',         ' t03p6 t15p6',
    ' t03p5 t15p5 t16p5',   ' t03p6 t15p6 t16p6',
    [1x24 char]             [1x24 char]
    ' t04p5 t10p5 t18p5',   ' t04p6 t10p6 t18p6',
    ' t05p5 t11p5 t19p5',   ' t05p6 t11p6 t19p6',
    ',,'                    ',,'
    ' t08p5 t13p5 t21p5',   ' t08p6 t13p6 t21p6',
    ' t07p5 t14p5 t22p5',   ' t07p6 t14p6 t22p6',
subAdjUse =
    1
    1
fndVactiveRow =
    6
    12
wUse =
Columns 1 through 5
    ' t06p1',    ' t06p2',    ' t06p3',    ' t06p4',    ' t06p5',
    ' t12p1',    ' t12p2',    ' t12p3',    ' t12p4',    ' t12p5',
Column 6
    ' t06p6',
    ' t12p6',
wRollUpStep1 =
Columns 1 through 3
    ' t06p1 t12p1',    ' t06p2 t12p2',    ' t06p3 t12p3',
Columns 4 through 6
    ' t06p4 t12p4',    ' t06p5 t12p5',    ' t06p6 t12p6',
wRollUpIncrement =
Columns 1 through 5
    ' t20p1',    ' t20p2',    ' t20p3',    ' t20p4',    ' t20p5',
Column 6
    ' t20p6',
wRollUpStep =
Columns 1 through 2
    ' t06p1 t12p1 t20p1',    ' t06p2 t12p2 t20p2',
Columns 3 through 4
    ' t06p3 t12p3 t20p3',    ' t06p4 t12p4 t20p4',
Columns 5 through 6
    ' t06p5 t12p5 t20p5',    ' t06p6 t12p6 t20p6',
wRollUp =
Columns 1 through 4
    {3x6 cell}    {6x6 cell}    {7x6 cell}    {4x6 cell}
Column 5

```

```

{1x6 cell}
wAccumulate =
Columns 1 through 2
    , ,
    , t01p1 t02p1'      , t01p2 t02p2'
    , t01p1 t03p1'      , t01p2 t03p2'
    , t01p1 t03p1 t04p1' , t01p2 t03p2 t04p2'
    , t01p1 t03p1 t05p1' , t01p2 t03p2 t05p2'
    , t03p1 t05p1 t06p1' , t03p2 t05p2 t06p2'
    , t01p1 t07p1'      , t01p2 t07p2'
    , t01p1 t07p1 t08p1' , t01p2 t07p2 t08p2'
    , t01p1 t02p1 t09p1' , t01p2 t02p2 t09p2'
    , t04p1 t10p1'      , t04p2 t10p2'
    , t05p1 t11p1'      , t05p2 t11p2'
    , t05p1 t06p1 t12p1' , t05p2 t06p2 t12p2'
    , t08p1 t13p1'      , t08p2 t13p2'
    , t07p1 t14p1'      , t07p2 t14p2'
    , t03p1 t15p1'      , t03p2 t15p2'
    , t03p1 t15p1 t16p1' , t03p2 t15p2 t16p2'
    [1x24 char]         [1x24 char]
    , t04p1 t10p1 t18p1' , t04p2 t10p2 t18p2'
    , t05p1 t11p1 t19p1' , t05p2 t11p2 t19p2'
    , t06p1 t12p1 t20p1' , t06p2 t12p2 t20p2'
    , t08p1 t13p1 t21p1' , t08p2 t13p2 t21p2'
    , t07p1 t14p1 t22p1' , t07p2 t14p2 t22p2'
Columns 3 through 4
    , ,
    , t01p3 t02p3'      , t01p4 t02p4'
    , t01p3 t03p3'      , t01p4 t03p4'
    , t01p3 t03p3 t04p3' , t01p4 t03p4 t04p4'
    , t01p3 t03p3 t05p3' , t01p4 t03p4 t05p4'
    , t03p3 t05p3 t06p3' , t03p4 t05p4 t06p4'
    , t01p3 t07p3'      , t01p4 t07p4'
    , t01p3 t07p3 t08p3' , t01p4 t07p4 t08p4'
    , t01p3 t02p3 t09p3' , t01p4 t02p4 t09p4'
    , t04p3 t10p3'      , t04p4 t10p4'
    , t05p3 t11p3'      , t05p4 t11p4'
    , t05p3 t06p3 t12p3' , t05p4 t06p4 t12p4'
    , t08p3 t13p3'      , t08p4 t13p4'
    , t07p3 t14p3'      , t07p4 t14p4'
    , t03p3 t15p3'      , t03p4 t15p4'
    , t03p3 t15p3 t16p3' , t03p4 t15p4 t16p4'
    [1x24 char]         [1x24 char]
    , t04p3 t10p3 t18p3' , t04p4 t10p4 t18p4'
    , t05p3 t11p3 t19p3' , t05p4 t11p4 t19p4'
    , t06p3 t12p3 t20p3' , t06p4 t12p4 t20p4'
    , t08p3 t13p3 t21p3' , t08p4 t13p4 t21p4'
    , t07p3 t14p3 t22p3' , t07p4 t14p4 t22p4'
Columns 5 through 6
    , ,
    , t01p5 t02p5'      , t01p6 t02p6'
    , t01p5 t03p5'      , t01p6 t03p6'
    , t01p5 t03p5 t04p5' , t01p6 t03p6 t04p6'
    , t01p5 t03p5 t05p5' , t01p6 t03p6 t05p6'
    , t03p5 t05p5 t06p5' , t03p6 t05p6 t06p6'
    , t01p5 t07p5'      , t01p6 t07p6'
    , t01p5 t07p5 t08p5' , t01p6 t07p6 t08p6'
    , t01p5 t02p5 t09p5' , t01p6 t02p6 t09p6'
    , t04p5 t10p5'      , t04p6 t10p6'
    , t05p5 t11p5'      , t05p6 t11p6'
    , t05p5 t06p5 t12p5' , t05p6 t06p6 t12p6'
    , t08p5 t13p5'      , t08p6 t13p6'
    , t07p5 t14p5'      , t07p6 t14p6'
    , t03p5 t15p5'      , t03p6 t15p6'
    , t03p5 t15p5 t16p5' , t03p6 t15p6 t16p6'
    [1x24 char]         [1x24 char]
    , t04p5 t10p5 t18p5' , t04p6 t10p6 t18p6'
    , t05p5 t11p5 t19p5' , t05p6 t11p6 t19p6'
    , t06p5 t12p5 t20p5' , t06p6 t12p6 t20p6'
    , t08p5 t13p5 t21p5' , t08p6 t13p6 t21p6'
    , t07p5 t14p5 t22p5' , t07p6 t14p6 t22p6'

```

K.10 Graph sub tree connectivity extraction

```

%.....
% testAdjMatrixExtract.m
% Test extract sub adjacency matrix from adjacency list
%.....
clc
clear all
format compact
% Output filename prefix
tgfFilePre='TSC'
% Pahl example - transposed
vertexLabels= { ...
    'a','b','c','d','e','f','g' ...
}
% Adjacency matrix
disp('Adjacency matrix:')
R=
    [0 0 0 0 0 0 0 ;...
     1 0 0 0 0 0 0 ;...
     1 0 0 1 0 0 0 ;...
     1 0 0 0 0 0 0 ;...
     0 1 0 1 0 0 0 ;...
     0 0 1 0 0 0 0 ;...
     0 0 0 1 1 1 0 ]'
% .tgf file output for yEd display input
isHomog=1
fileTC=horzcat(tgfFilePre,'TreeConnectivity',' .tgf')
% output adjacency matrix
tgfWrite(fileTC,R,isHomog,vertexLabels,{});
% form adjacency list
disp('Adjacency list:')
[adjListR,nrowLR,ncolLR] = AdjacencyList(R)
% extract sub adjacency matrices from adjacency lists
for nvertex=1:size(R,2)
    [adjMatrixR,vactive] = adjListExtract(adjListR,nvertex)
    fileOut=horzcat(tgfFilePre,'TreeSubConnec',num2str(nvertex),' .tgf')
    % set up vertex labels for graph data output file
    vertexLabelsOut=vertexLabels(find(vactive))
    tgfWrite(fileOut,adjMatrixR,isHomog,vertexLabelsOut,{});
    [adjsubListR,nrowLR,ncolLR] = AdjacencyList(adjMatrixR)
end
vertexList=[3,4]
[adjMatrix,vactive] = adjListMultExtract(adjListR,vertexList)

```

```

tgfFilePre =
TSC
vertexLabels =
    'a'    'b'    'c'    'd'    'e'    'f'    'g'
Adjacency matrix:
R =
     0     1     1     1     0     0     0
     0     0     0     0     1     0     0
     0     0     0     0     0     1     0
     0     0     1     0     1     0     1
     0     0     0     0     0     0     1
     0     0     0     0     0     0     1
     0     0     0     0     0     0     0
isHomog =
     1
fileTC =
TSCTreeConnectivity.tgf
File TSCTreeConnectivity.tgf opened
nRows =
     7
nCols =
     7
File TSCTreeConnectivity.tgf closed
Adjacency list:

```

```

adjListR =
  1  2  3  4
  2  5  0  0
  3  6  0  0
  4  3  5  7
  5  7  0  0
  6  7  0  0
  7  0  0  0
nrowLR =
  7
ncollLR =
  4
adjMatrixR =
  []
vactive =
  0  0  0  0  0  0  0
fileOut =
  TSCTreeSubConnec1.tgf
vertexLabelsOut =
  Empty cell array: 1-by-0
File TSCTreeSubConnec1.tgf opened
nRows =
  0
nCols =
  0
File TSCTreeSubConnec1.tgf closed
adjsubListR =
  []
nrowLR =
  0
ncollLR =
  0
adjMatrixR =
  0  1
  0  0
vactive =
  1  1  0  0  0  0  0
fileOut =
  TSCTreeSubConnec2.tgf
vertexLabelsOut =
  'a'  'b'
File TSCTreeSubConnec2.tgf opened
nRows =
  2
nCols =
  2
File TSCTreeSubConnec2.tgf closed
adjsubListR =
  1  2
  2  0
nrowLR =
  2
ncollLR =
  2
adjMatrixR =
  0  1  0
  0  0  0
  0  1  0
vactive =
  1  0  1  1  0  0  0
fileOut =
  TSCTreeSubConnec3.tgf
vertexLabelsOut =
  'a'  'c'  'd'
File TSCTreeSubConnec3.tgf opened
nRows =
  3
nCols =
  3
File TSCTreeSubConnec3.tgf closed
adjsubListR =

```

```

1      2
2      0
3      2
nrowLR =
3
ncollLR =
2
adjMatrixR =
0      1
0      0
vactive =
1      0      0      1      0      0      0
fileOut =
TSCTreeSubConnec4.tgf
vertexLabelsOut =
'a'      'd'
File TSCTreeSubConnec4.tgf opened
nRows =
2
nCols =
2
File TSCTreeSubConnec4.tgf closed
adjsubListR =
1      2
2      0
nrowLR =
2
ncollLR =
2
adjMatrixR =
0      0      1
0      0      1
0      0      0
vactive =
0      1      0      1      1      0      0
fileOut =
TSCTreeSubConnec5.tgf
vertexLabelsOut =
'b'      'd'      'e'
File TSCTreeSubConnec5.tgf opened
nRows =
3
nCols =
3
File TSCTreeSubConnec5.tgf closed
adjsubListR =
1      3
2      3
3      0
nrowLR =
3
ncollLR =
2
adjMatrixR =
0      1
0      0
vactive =
0      0      1      0      0      1      0
fileOut =
TSCTreeSubConnec6.tgf
vertexLabelsOut =
'c'      'f'
File TSCTreeSubConnec6.tgf opened
nRows =
2
nCols =
2
File TSCTreeSubConnec6.tgf closed
adjsubListR =
1      2
2      0

```



```

nrowLR =
    2
ncollR =
    2
adjMatrixR =
    0    0    0    1
    0    0    0    1
    0    0    0    1
    0    0    0    0
vactive =
    0    0    0    1    1    1    1
fileOut =
    TSCTreeSubConnec7.tgf
vertexLabelsOut =
    'd'    'e'    'f'    'g'
File TSCTreeSubConnec7.tgf opened
nRows =
    4
nCols =
    4
File TSCTreeSubConnec7.tgf closed
adjsubListR =
    1    4
    2    4
    3    4
    4    0
nrowLR =
    4
ncollR =
    2
vertexList =
    3    4
adjMatrix =
    0    1    1
    0    0    0
    0    1    0
vactive =
    1    0    1    1    0    0    0

```

K.11 Multiple sub tree connectivity extraction

```

%.....
% testAdjmatrixMultExtract.m
% Test extraction of adjacency matrix from adjacency list
% given multiple vertices
%.....
clc
clear all
format compact
% Output filename prefix
tgfFilePre='TSC'
% Pahl example - transposed
vertexLabels= { ...
    'a','b','c','d','e','f','g' ...
}
% Adjacency matrix
disp('Adjacency matrix:')
R= [0 0 0 0 0 0 0 ;...
    1 0 0 0 0 0 0 ;...
    1 0 0 1 0 0 0 ;...
    1 0 0 0 0 0 0 ;...
    0 1 0 1 0 0 0 ;...
    0 0 1 0 0 0 0 ;...
    0 0 0 1 1 1 0 ]'
% form adjacency list
disp('Adjacency list:')
[adjListR,nrowLR,ncollR] = AdjacencyList(R)
% select vertices for extraction

```

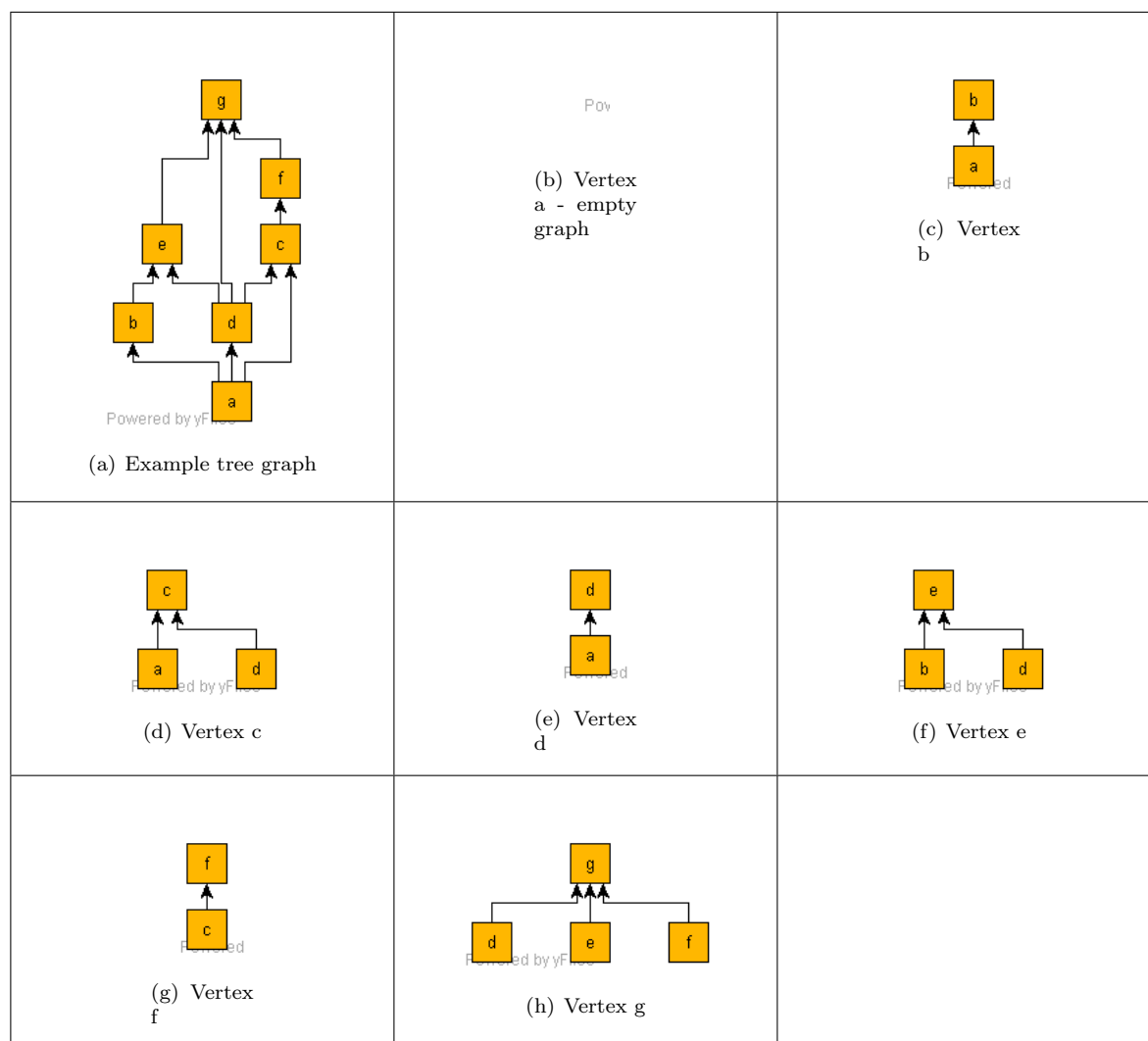


Figure K.9: Sub-tree connectivity extraction per vertex as listed: (a) Example tree graph; (b) Vertex a - empty graph; (c) Vertex b; (d) Vertex c; (e) Vertex d; (f) Vertex e; (g) Vertex f; (h) Vertex g;

```
disp('Vertices selected for extraction:')
vertexList=[1,5,7]
disp('Compute compound adjacency matrix with active vertices:')
[adjMatrix,vactive] = adjListMultExtract(adjListR,vertexList)
% list active vertex labels
disp('Active vertex labels :')
vertexLabelsActive=vertexLabels(find(vactive))
```

```
tgfFilePre =
TSC
vertexLabels =
'a' 'b' 'c' 'd' 'e' 'f' 'g'
Adjacency matrix:
R =
0 1 1 1 0 0 0
0 0 0 0 1 0 0
0 0 0 0 0 1 0
0 0 1 0 1 0 1
0 0 0 0 0 0 1
0 0 0 0 0 0 1
0 0 0 0 0 0 0
```

```
Adjacency list:
adjListR =
    1    2    3    4
    2    5    0    0
    3    6    0    0
    4    3    5    7
    5    7    0    0
    6    7    0    0
    7    0    0    0
nrowLR =
    7
ncolLR =
    4
Vertices selected for extraction:
vertexList =
    1    5    7
Compute compound adjacency matrix with active vertices:
adjMatrix =
    0    0    1    0    0
    0    0    1    0    1
    0    0    0    0    1
    0    0    0    0    1
    0    0    0    0    0
vactive =
    0    1    0    1    1    1    1
Active vertex labels :
vertexLabelsActive =
    'b'    'd'    'e'    'f'    'g'
```

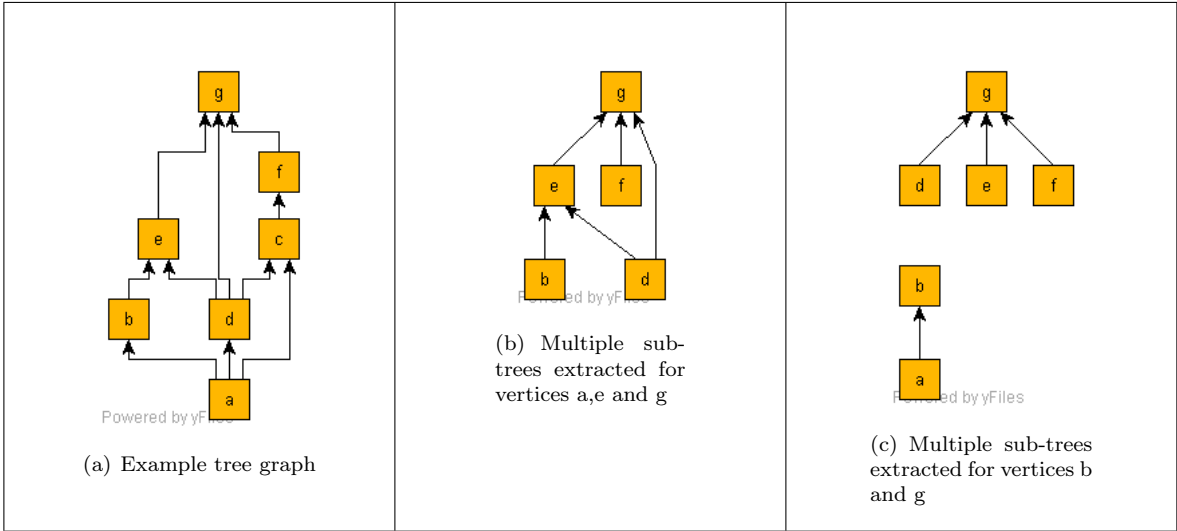


Figure K.10: Sub-tree extraction

K.12 Sub-tree extraction MATLAB functions

The MATLAB functions are used in the examples shown in sections K.10 and K.11.

```
function [AdjList,nrowL,ncolL] = AdjacencyList(AdjMatrix)
%.....
% Generate adjacency list given adjacency matrix
% Adjacency list in matrix format – ignore 0 entries
%.....
AdjList = [];
nrowA=size(AdjMatrix,1);
```

```

ncolA=size ( AdjMatrix ,2);
for ir=1:nrowA
    irowL=ir ;
    AdjList(irowL,1)=ir ;
    icolL=1;
    for ic=1:ncolA
        if ( AdjMatrix( ir , ic)==1)
            icolL=icolL+1;
            AdjList( irowL , icolL)=ic ;
        end
    end
end
nrowL=size ( AdjList ,1);
ncolL=size ( AdjList ,2);

```

```

function [adjMatrix,vactive] = adjListMultExtract(adjList,nvertexList)
% .....
% function adjListMultExtract.m
% Extract all edges linked to a vertex and return
% adjMatrix – subgraph in adjacency matrix format
% vactive – Boolean list of active vertices
% can be converted to list format if required
% nvertexList – array of vertex numbers to process
% .....
% limits
nVertices=max( size( adjList ,1),max(max( adjList )));
% set up blank adjacency matrix
adjMatrixInterim=zeros(nVertices);
vactive=zeros(1,nVertices);
for nvL=1:size(nvertexList,2)
    nvertex=nvertexList(nvL);
    if not((nvertex>nVertices))
% set limits to active vertex numbers
for nv=1:size(adjList,1)
    nEdgesRow=size( adjList(nv,:) ,2);
    for ne=2:nEdgesRow
% end vertex of edge
        mv=adjList(nv,ne);
% skip zero entries as well as entries not linked to selected vertex
        if not(mv==0)&& (mv==nvertex)
            adjMatrixInterim(nv,mv)=1;
% keep track of active vertex entries
            vactive(nv)=1;
            vactive(mv)=1;
        end
    end
end
end
end
% disp(['vactive: ',num2str(vactive)])
% adjMatrixInterim
% Add active columns to output matrix
adjCols=[];
for iv=1:nVertices
    if (vactive(iv)==1)
        adjCols=[adjCols,adjMatrixInterim(:,iv)];
    end
end
% Add active rows to output matrix
% adjCols
adjMatrix=[];
for iv=1:nVertices
    if (vactive(iv)==1)
        adjMatrix=[adjMatrix;adjCols(iv,:)];
    end
end
end
end

```

```

function [adjMatrix,vactive] = adjListExtract(adjList,nvertex)
%.....
% Extract all edges linked to a vertex and return
% adjMatrix - subgraph in adjacency matrix format
% vactive - Boolean list of active vertices
% can be converted to list format if required
% nvertex - vertex number to process
%.....
% initialise output
adjMatrix=[];
nVertices=max(size(adjList,1),max(max(adjList)));
if not((nvertex>nVertices))
% set up blank adjacency matrix
adjMatrixInterim=zeros(nVertices);
vactive=zeros(1,nVertices);
% set limits to active vertex numbers
for nv=1:size(adjList,1)
    nEdgesRow=size(adjList(nv,:),2);
    for ne=2:nEdgesRow
% end vertex of edge
        mv=adjList(nv,ne);
% skip zero entries as well as entries not linked to selected vertex
        if not(mv==0)&& (mv==nvertex)
            adjMatrixInterim(nv,mv)=1;
% keep track of active vertex entries
            vactive(nv)=1;
            vactive(mv)=1;
        end
    end
end
end
% disp(['vactive: ',num2str(vactive)])
% adjMatrixInterim
% Add active columns to output matrix
adjCols=[];
for iv=1:nVertices
    if (vactive(iv)==1)
        adjCols=[adjCols,adjMatrixInterim(:,iv)];
    end
end
% Add active rows to output matrix
% adjCols
for iv=1:nVertices
    if (vactive(iv)==1)
        adjMatrix=[adjMatrix;adjCols(iv,:)];
    end
end
end
end

```

K.13 File format for yEd graph display program (.tgf)

```

1 a
2 b
3 c
4 d
5 e
6 f
7 g
#
1 2
1 3
2 4
2 5
3 6
3 7

```

Appendix L

Marketing Management - Sample Documents

L.1 Marketing activity planning and status reporting

Figure L.1 shows a sample marketing activity planning sheet and figure L.2 shows the format of a marketing action status report.

L.2 Marketing budgeting and income budget planning

Figure L.3 shows a sample marketing budget report which is used with probability estimates to forecast business fee income.

L.3 Marketing budget according to project status

Figure L.4 taken from Puttergill [99] shows a typical project report indicating project marketing status.

BEMARKINGSAKSIE VIR AANDEELHOUSERS GEUSTYN FORSYTH & JOUBERT - STELLENBOSCH				
NAAM: <i>B. Strasheim</i>		PERIODE: 14/2/89 - 28/03/89		
AKTIWITEIT	GEWIG	DOELWIT	DOELWITBEVREDIGING	
1. WELWILLENDHEIDSBESOEKE				
1.1 Besoek potensiële kliënte	2			
1.2 Besoek gewese kliënte	2			
1.3 Besoek bestaande kliënte	1	Stad Welkom	1	
2. OPENBARE BLOOTSTELLING				
2.1 Akademiese publikasie	9			
2.2 Referaatlewing/ praatjies	8			
2.3 Berig/nuusitem/ persvrystelling	7			
2.4 Heraanbieding/verwerking van vorige publikasie	5			
3. VERENIGINGS/INSTITUTE (Tegnies)				
3.1 Deelname aan komitees/ bestuur	8	SMSI Rekenaar- komitee	8	
3.2 Bywoning van simposiums/ kongresse	4	CONSAS Sewane Gossie 13/02/89 (Londen)	(2)	✓
3.3 Bywoning van vergaderings	1	SMSI Rekenaar komitee	2	
4. ONTHAAL / BYEENKOMS				
4.1 Groter skaal georgani- seerd	3	Plaaslike/Nas- ionale		
4.2 Klein groepie - georgani- seer	2			
4.3 Privaat man tot man	1	Munisipale Raad Stellenbosch Tegnies	1	
5. ADMINISTRATIEF				
5.1 Opdatering kliënte-paneel	6			
5.2 Guns bewys aan kliënt: bv. opdateer Spesifikasie	4			
5.3 Organogramme van kliënte bekom en onderhou	2	Stad Kempton- Park Stad Welkom	4	
5.4 Kliëntedatabasis opstel/ in stand hou.	1	HP Steenkamp en Vind	1	
5.5 Nuusmanwerk	1			
TOTALE PUNT		XXXXXXXXXXXXXXXXXXXXXXX	12	XXXXXXXXXXXXXXXXXXXXXXX

Figure L.1: GFJ Inc - Marketing activity planning sheet

[illegible]

Figure L.2: *GFJ Inc. - Sample marketing management action status report*

G F & J Suidelike Streek (CS) - Stellenbosch (STL) & George (GEG) Kantore																				
=====																				
Bemarkingsstatusomring: 1990-08-20																				
=====																				
Klient	Projek	Projekteier	Afdeling	Kapitaal	GFAJ Inkomste	Bepian	Datum Konstruksie %	Status	90-08	90-09	90-10	90-11	90-12	91-01	91-02	91-03	91-07	91-08	91-09	Kontrole

F du Toit	Watersinities	DuToitAJ	CIVIN	R6 000 000	R200 000	90-11	91-03	33				R13 200	R13 200	R13 200	R13 200	R13 200				R200 000
Brandtcrus	Saldanha_SAD	DuToitAJ	CIVIN		R10 000	90-08	90-08	100	R10 000											R10 000
Westerland	Uniepark	DuToitAJ	CIVIN		R30 000	90-10	91-02	33												R50 000
R B M	RBN Water	StrasheimAVB	CIVIN	R10 000 000	R10 000 000	90-09	91-06	25		R2 500										R10 000
Ling-les	Meesterplan	StrasheimAVB	CIVIN/VEIN		R150 000	90-11	91-05	60			R11 250	R11 250	R11 250	R11 250	R11 250	R11 250				R150 000
Strad Welton	Bestuursplanne	StrasheimAVB	CIVIN/VEIN		R500 000	90-11	91-05	50		R9 375	R9 375	R9 375	R9 375	R9 375	R9 375	R9 375				R500 000
UK-SOR	Watersvallei F2	DuToitAJ	CIVIN	R9 000 000	R100 000	90-12	91-05	25												R100 000
DeLaFontaine	Horwood Persael	Ridgway	CIVIN	R1 000 000	R100 000	90-12	91-05	25												R300 000
A B Prinsloo	Teem-die-Bult	KrigeAJ	CIVIN	R3 000 000	R300 000	90-09	91-03	25												R100 000
UK-SOR	Scottsdale	KrigeAJ	CIVIN	R1 000 000	R100 000	90-10	91-05	90												R200 000
Ling-les	Ling-les Randdop	KrigeAJ	CIVIN	R8 000 000	R700 000	90-10	91-05	75												R700 000
DeLaFontaine	Bellor Dienste	KrigeAJ	CIVIN	R2 000 000	R200 000	91-01	91-06	25												R200 000
Mun GBR	Riolering	KrigeAJ	CIVIN	R1 400 000	R150 000	90-11	91-04	50												R150 000
UK-SOR	SLP Platters	PollardST	CIVIN	R1 000 000	R700 000	90-09	90-11	45												R700 000
UK-SOR	SLP Riol	PollardST	CIVIN	R300 000	R30 000	90-11	91-03	25												R300 000
UK-SOR	SLP Infrastruk.	PollardST	CIVIN	R2 000 000	R180 000	91-01	91-07	25												R200 000
Mun GBR	Rioolpompstasie	PollardST	CIVIN	R1 400 000	R160 000	91-05	91-05	50												R160 000
Subtotal CIVIN				R46 300 000	R3 065 000			49	R10 000	R23 714	R114 514	R172 857	R196 324	R181 286	R244 286	R177 986	R6 429	R0	R46 500	R3 065 000
Dept Onderwys	Handle Skool	Ridgway	STRUC	R5 000 000	R200 000	90-12	91-07	45									R11 250			R200 000
Subtotal STRUC				R3 000 000	R200 000			45												R200 000
Munisip Ceres	Casoor/loopmoed	BosmaDE	WATSP		R8 000	90-08	91-06	45	R327	R327	R327	R327	R327	R327	R327	R327				R8 000
Munisip Ceres	Toegangspad	BosmaDE	WATSP		R1 000 000	90-11	91-05	45												R120 000
Subtotal WATSP				R1 000 000	R128 000			45	R327	R327	R327	R327	R327	R327	R327	R327	R0	R0	R0	R128 000
Munisip Plett	Vaste Afval	Palmig	WASTE		R6 000	90-12		25												R6 000
Paarl UK-SOR	Vaste Afval	Palmig	WASTE		R20 000	91-02		25												R20 000
Ling-les	Vaste Afval	Palmig	WASTE		R150 000	90-11	91-02	90												R150 000
Subtotal WASTE				R2 000 000	R176 000			80	R0	R0	R0	R0	R0	R0	R0	R0	R0	R0	R0	R176 000
Mun GBR	Rioolwerke	Palmig	WATRT		R1 000 000	91-02	91-07	50												R120 000
Mun Porterville	Rioolwerke Opgrd	Palmig	WATRT		R500 000	91-01	91-05	25												R42 275
Subtotal WATRT				R1 500 000	R162 275			43	R0	R0	R0	R0	R0	R0	R0	R0	R2 114	R2 114	R19 614	R162 275
Opkomring vir STL kantoor																				R3 731 275
Rioolwerke Voorspeller:																				
Uitsgesluit:	KPR projekte																			
GGG Projekte	GGG Projekte																			

Opkomring vir STL kantoor																				
Rioolwerke Voorspeller:																				
R10 327 R24 042 R114 842 R261 899 R216 045 R189 327 R307 827 R188 527																				
e Bemarkingsstatus %																				
R8 542 R2 114 R66 114 R1 759 919																				
47																				

Figure L.3: *GFJ Inc. - Sample marketing management budget derived from control report*

TABLE 1A BUDGET 1985/6 XYZ COMPANY DETAILED BUDGET COMPARISON		TOTAL COMPANY		NOTES				
		1985 BUDGET	DEC 84 Y-T-D ANNUALIZED	1984 BUDGET	1983 ACTUAL			
FEES								
- In Hand	22,100,000	45.0%	22,007,000	62.4%	24,600,000	66.1%	20,500,000	56.3%
- In View	15,019,000	30.6%	12,000,000	34.0%	10,145,000	27.3%	5,200,000	14.3%
- To Be Obtained	12,000,000	24.4%	1,250,000	3.5%	2,459,000	6.6%	10,700,000	29.4%
Subtotal	49,119,000	100.0%	35,257,000	100.0%	37,204,000	100.0%	36,400,000	100.0%
Management Reserve On Fees	0		0	0		0		
Net Total Fees	49,119,000		35,257,000	37,204,000		36,400,000		
Joint Venture Income	0		421,000	352,000		0		
Construction Income	0		646,000	850,000		0		
TOTAL FEES & OTHER INCOME	49,119,000	239.8%	36,324,000	274.1%	38,406,000	274.3%	36,400,000	187.7%
PAYROLL COST OF SERVICES	20,486,000	100.0%	13,250,000	100.0%	14,000,000	100.0%	19,396,000	100.0%
GROSS PROFIT	28,633,000	139.8%	23,074,000	174.1%	24,406,000	174.3%	17,004,000	87.7%
Corporate reserves	574,000		0	0		0		
ADJUSTED GROSS PROFIT	28,059,000	137.0%	23,074,000	174.1%	24,406,000	174.3%	17,004,000	87.7%
OPERATING EXPENSES (Overhead)								
Business Development	4,196,000	20.5%	3,200,000	24.2%	2,500,000	17.9%	1,501,000	7.7%
Unassigned Technical Time	2,707,000	13.2%	47,000	0.4%	1,267,000	9.1%	26,000	0.1%
Administration	8,892,000	43.4%	8,245,000	62.2%	9,947,000	71.1%	6,446,000	33.2%
Premises	6,358,000	31.0%	4,056,000	30.6%	4,012,000	28.7%	3,817,000	19.7%
Facilities	(376,000)	-1.8%	(28,928)	-0.2%	69,100	0.5%	480,000	2.5%
Financial Charges	2,236,000	10.9%	1,737,000	13.1%	1,031,000	7.4%	3,322,000	17.1%
Corporate Control	668,000	3.3%	0	0.0%	0	0.0%	0	0.0%
Subtotal	24,681,000	120.5%	17,256,072	130.2%	18,826,100	134.5%	15,592,000	80.4%
Corp. Reserve On Overhead	600,000		0	0		0		
TOTAL OVERHEAD	25,281,000	123.4%	17,256,072	130.2%	18,826,100	134.5%	15,592,000	80.4%
OPERATING PROFIT	2,778,000	13.6%	5,817,928	43.9%	5,579,900	39.9%	1,412,000	7.3%
Foreign Exchange Adjustments	0		0	0		0		
Incentive	138,000		0	250,000		0		
Tax provision	0		0	0		860,000		
NET INCOME (Operations)	2,640,000	12.9%	5,817,928	43.9%	5,329,900	38.1%	552,000	2.8%
Investment Income	120,000		120,000	120,000		120,000		
NET INCOME	2,760,000	13.5%	5,937,928	44.8%	5,449,900	38.9%	672,000	3.5%

1. 1983 REFERS TO FINANCIAL YEAR 1983/4
1985 REFERS TO FINANCIAL YEAR 1985/6

2. THE FIGURES INCLUDED IN THIS REPORT
ARE HYPOTHETICAL AND ARE NOT INTENDED
TO REFLECT A REAL COMPANY.

3. DEC 84 Y-T-D ANNUALIZED ARE THE ACTUAL
RESULTS FOR THE YEAR TO DATE TO
DECEMBER 1984 PROJECTED TO YEAR
END FEBRUARY 1985

Figure L.4: Puttergill - Sample corporate budget showing FEE income as per project marketing classification

Bibliography and References

- [1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundation of Databases*. Addison-Wesley Publishing Company, 1995. ISBN 0-201-53771-0.
- [2] P. Albertyn and D. A. Fourie. Computerisation of plan office data: Directorate of National Roads. In *Annual Conference on Computers in Civil Engineering – Computers 86*, Apr 1986. ISBN 0-7988-3692-2.
- [3] Sinan Si Alhir. *UML in a nutshell - A Desktop Quick Reference*. O'Reilly, 1998. ISBN 1-56592-448-7.
- [4] Altova GmbH / Altova, Inc. XMLSpy - XML editor for modelling, editing, transforming, and debugging XML technologies, 2006. URL http://www.altova.com/products/xmlspy/xml_editor.html.
- [5] Alvin A. Arens and James K. Loebbecke. *Auditing: An Integrated Approach*. Prentice-Hall Inc., Third edition, 1984. ISBN 0-13-051749-6.
- [6] W. Armstrong. Dependency Structures of Database Relationships. In *Proceedings of the IFIP Congress*, 1974.
- [7] Ross Ashby. *Introduction to Cybernetics*. Chapman & Hall, 1964.
- [8] Erik Aslaksen and Rod Belcher. *Systems Engineering*. Prentice Hall, 1991. ISBN 0-13-880402-8.
- [9] Paolo Atzeni and Valeria De Antonellis. *Relational Database Theory*. The Benjamin/Cummings Publishing Company, Inc., 1993. ISBN 0-8053-0249-2.
- [10] Azzurri Limited. Database Modelling in Eclipse, 2006. URL <http://www.azzurri.jp/en/software/clay/>.
- [11] V. K. Balakrishnan. *Schaum's Outline of Theory and Problems of Graph Theory*. Schaum's Outline Series McGraw-Hill, 1997. ISBN 0-07-005489-4.
- [12] Kenneth J. Barlow. *Professional Management for Consulting Engineers Volume I & II*. Kenneth J. Barlow Limited, Toronto, Ontario, 1972.
- [13] Michael Barr. Introduction to closed-loop control, 2002. URL <http://www.netrino.com/Publications/Glossary/PID.html>.
- [14] John Beishon and Technology Foundation Course Team. *Systems*. The Open University Press, 1971. ISBN 335 02500 5.
- [15] Simon Bennett, John Skelton, and Ken Lunn. *Schaum's Outline of UML*. McGraw-Hill, 2001. ISBN 0-07-709673-8.
- [16] Ivan Bester and R.J. Koch. Risk Management (Risikobestuur). Course Notes, University of Stellenbosch Business School, 1984.
- [17] Prof. Carl Beucke. Computer Aided Drafting (CAD) Technology Course. Presented at the Department of Civil Engineering at the University of Stellenbosch, July 2002.
- [18] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modelling Language User Guide*. The Addison-Wesley Object Technology Series. Addison-Wesley, Rational Software Corporation, 1999. ISBN 0201571684.

- [19] Kenneth E. Boulding. General Systems Theory - The Skeleton of Science. *Management Science*, 2 (3):197–208, April 1956.
- [20] Cobus Burgers. Supporting the Project Office through document management. In *SAICE Division of Information Technology Twenty-fifth Annual Symposium – Is IT Sustainable in Engineering?*, Sep 2003.
- [21] Rory Burke. *Project Management Planning & Control Techniques*. Promatec International, Stratford Upon Avon, Cape Town, third edition, 1999. ISBN 0-620-23414-8.
- [22] William L. Chapman, A. Terry Bahill, and A. Wayne Wymore. *Engineering Modelling and Design*. CRC Press, 1992. ISBN 0-8493-8011-1.
- [23] Gary Chartrand and Ortrud R. Oellerman. *Applied and Algorithmic Graph Theory*. McGraw-Hill, Inc., 1993.
- [24] E. Codd. A Relational Model for Large Shared Databanks. In *CACM*, volume 37:6, June 1970.
- [25] Corel Corporation. Corel WordPerfect Office X3 - Professional Edition - Paradox, 2006. URL <http://www.corel.com>.
- [26] Mercia Cronje. Engineering process model: Detection of cycles and solution of equations. Master's thesis, University of Stellenbosch, April 2006.
- [27] C. J. Date. *Introduction to Database Systems , Seventh Edition*. Addison-Wesley, 2000. ISBN 0-201-38590-2.
- [28] Dr. Wim De Villiers. *Die Aspekte van Bestuur – Gencor Beperk Bestuurshandleidings*. Gencor Beperk, 1979.
- [29] Deltek Systems Inc. Deltek software, 2006. URL <http://www.deltek.com/>.
- [30] Klaus R. Dittrich, Umeshwar Dayal, and Alejandro P. Buchmann, editors. *On Object-Oriented Database Systems*. Topics in Information Systems. Springer-Verlag, 1991. ISBN 3-540-53496-2.
- [31] Decision Processes International (DPI). Decision Processes International - Leader in Critical Thinking, 2006. URL <http://www.decisionprocesses.com>.
- [32] P. G. Du Plessis, editor. *Applied Business Management*. Kagiso Tertiary, 1996. ISBN 0-7986-3567-3.
- [33] P. G. Du Plessis, editor. *Toegepaste Ondernemingsbestuur*. Kagiso Tertiary, 1996. ISBN 0-7986-3566-5.
- [34] Alan F. Dutka and Howard H. Hanson. *Fundamentals of Data Normalization*. Addison-Wesley Publishing Company, Boston, MA, USA, 1989. ISBN 0-201-06645-9.
- [35] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems Third Edition*. Addison-Wesley, 2000. ISBN 0-201-54263-3.
- [36] F.E. Emery and E.L. Trist. Socio-technical systems. *Management Science, Models and Techniques*, 2:83–97, 1960.
- [37] A.B. Eygelaar and G.C. Van Rooyen. Optimal Scheduling Of Activities In An Engineering Project. In *SAISC Steel Conference, Johannesburg*, 2006.
- [38] Anton Burger Eygelaar. Modelling the Engineering Process. Technical Report I01/2004, University of Stellenbosch, Department of Civil Engineering, November 2004.
- [39] Len Fertuck. *System Analysis and Design with Modern Methods*. Business and Education Technologies, 1995. ISBN 0-697-16218-4.
- [40] Ben Forta. *SQL in 10 Minutes*. SAMS Publishing, 2004. ISBN 0-672-32567-5.
- [41] P. D. Gerber, P. S. Nel, and P. S. Van Dyk. *Menslike Hulpbron Bestuur*. International Thompson Publishing (Southern Africa) (Pty) Ltd, 1998. ISBN 1 86864 071 X.

- [42] Ernest Glad and Hugh Becker. *Activity-Based Costing and Management*. Juta & Company Ltd., 1994. ISBN 0 7021 2792 2.
- [43] Hassan Gomaa. *Designing Concurrent Distributed, and Real-Time Applications with UML*. The Addison-Wesley Object Technology Series. Addison-Wesley, 2000. ISBN 0-201-65793-7.
- [44] Google Inc. Google and Google South Africa, 2006. URL <http://www.google.com>.
- [45] Google Inc. Google Images, 2006. URL <http://images.google.com>.
- [46] Michel Goossens, Frank Mittlebach, and Alexander Samarin. *The L^AT_EX Companion*. Addison-Wesley Series on Tools and Techniques for Computer Typesetting. Addison-Wesley, 1994. ISBN 0-201-54199-8.
- [47] South African Government. *Engineering Professions Act, Act No. 46 of 2000*. South African Government, 2000.
- [48] Jeanette Greeff. Derivation and Implementation of an Extended Entity-Relationship Data Model. Master of science, University of Stellenbosch Department of Computer Science, December 1990. Supervisor M H Rennhackkamp.
- [49] Martin Green. Office tips from Martin Green. World Wide Web, 2006. URL <http://www.fontstuff.com/access>.
- [50] Arno Greyling. PROMAN Practice Management System, 2006. URL <http://www.akron.co.za/>.
- [51] Johnathan Gross and Jay Yellen. *Graph Theory and its Applications*. CRC Press, 1999. ISBN 0-8493-3982-0.
- [52] Johnathan Gross and Jay Yellen, editors. *Handbook of Graph Theory*. CRC Press, 2004. ISBN 1-58488-090-2.
- [53] Duane Hanselman and Bruce Littlefield. *The Student Edition of MATLAB*. The MATLAB Curriculum Series. Prentice Hall, 1997. ISBN 0-13-272550-9.
- [54] Paul Helman. *The Science of Database Management*. Richard D. Irwin Inc., 1994. ISBN 0-256-13438-3 ISBN 0-256-15881-9 (Intl. Ed.).
- [55] Henk B. Heymans. *The Business Approach to Auditing*. Juta & Company Ltd., revision service 10, 2003 edition, 1983-2006. ISBN 0 7021 1428 6.
- [56] Forrest Houlette. *SQL A Beginner's Guide*. Osborne/ McGraw-Hill, 2001. ISBN 0-07-213096-2.
- [57] W. Huhnt. Process Models as a Base for the Co-ordination of Construction Projects. In *International Conference on Computing in Civil and Building Engineering*, number 9th in International Conference on Computing in Civil and Building Engineering, Taipei, Taiwan, April 2002.
- [58] Wolfgang Huhnt. Consistent Models for Controlling Technical and Economic Processes. In *International Conference on Computing in Civil and Building Engineering*, number 8th in International Conference on Computing in Civil and Building Engineering, Stanford, USA, 2000.
- [59] Wolfgang Huhnt. Process Modelling Basics. Technical Report Version 1.0 22.12.2004 11:55, Technical University of Berlin, 2004.
- [60] Wolfgang Huhnt. Progress Measurement in Planning Processes on the Base of Process Models. In *Xth International Conference on Computing in Civil and Building Engineering, Weimar, Germany*, June 2004.
- [61] Wolfgang Huhnt. Modelling Planning Processes. Technical Report Version 1.0, 04.01.2005 14:05, Technical University of Berlin, 2004.
- [62] Wolfgang Huhnt, Michael Kluge, and Hans-Jürgen Laufer. Mapping Technical Processes into Standard Software for Business Support. In *Construction Information Technology (CIT)*, Construction Information Technology (CIT), Reikjavik, Iceland, 2000.

- [63] Keith Huxam and Phillip Haupt. *South African VAT The Book*. H&H Publications and Hedron Tax Consulting and Publishing CC, 1991. ISBN 0-9583112-5-0.
- [64] International Facilities Management Association IFMA. International facilities management, 2006. URL <http://www.ifma.org/>.
- [65] JabRef. JabRef open source Bibliography Reference Manager, 2006. URL <http://jabref.sourceforge.net/>.
- [66] J.C. Jones and Edited by Singleton, W.T. et al. *The design of man-machine systems in The human operator in complex systems*. Taylor & Francis, 1967.
- [67] Michail Kagioglou, Rachel Cooper, Ghassan Aouad, and Martin Sexton. Rethinking construction: the Generic Design and Construction Process Protocol. *Engineering Construction & Architectural Management*, 7 Issue 2:141, June 2000.
- [68] Won Kim and Frederick H. Lochovsky, editors. *Object-Oriented Concepts, Databases and Applications*. ACM Press Frontier Series. ACM Press Addison-Wesley Publishing Company, 1989. ISBN:0-201-14410-7.
- [69] Helmut Kopka and Patrick W. Daly. *Guide to L^AT_EX*. Addison-Wesley Series on Tools and Techniques for Computer Typesetting. Addison-Wesley, fourth edition, 2003. ISBN 0-321-17385-6.
- [70] Leslie Lamport. *L^AT_EX Document Preparation System User's Guide And Reference Manual*. Addison-Wesley Series on Tools and Techniques for Computer Typesetting. Addison - Wesley, second edition, 1994. ISBN 0-201-52983-1.
- [71] Jeff Lawrence. Computational management techniques for project management. In *SAICE Division of Information Technology Twenty-fifth Annual Symposium - Is IT Sustainable in Engineering?*, Sep 2003.
- [72] Philip M. Lewis, Arthur Bernstein, and Michael Kifer. *Databases and Transaction Processing*. Addison-Wesley, first edition, 2002. ISBN 0-201-70872-8.
- [73] Seymour Lipschutz. *Set Theory and Related Topics*. Schaum's Outline Series - McGraw-Hill Book Company, 1964. ISBN 07-037986-6.
- [74] Seymour Lipschutz. *Schaum's Outline of Essential Computer Mathematics*. Schaum's Outline Series - McGraw-Hill Book Company, 1982. ISBN 0-07-037990-4.
- [75] Paul Litwin, Ken Getz, and Mike Gilbert. *Access 97 Developer's Handbook*. Sybex, third edition, 1997. ISBN 0-7821-1941-7.
- [76] Paul Litwin, Ken Getz, and Mike Gilbert. *Access 2002 Desktop Developer's Handbook*. Sybex, 2001. ISBN 0-7821-4009-2.
- [77] Paul Litwin, Ken Getz, and Mike Gunderloy. *Access 2002 Enterprise Developer's Handbook*. Sybex, 2001. ISBN 0-7821-4010-6.
- [78] Guy Macleod. *Starting Your Own Business in South Africa*. Oxford University Press, sixth edition, 1983. ISBN 0 19 570558 0.
- [79] David Maier. *The Theory of Relational Databases*. Computer Science Press, 1983. ISBN 0-914894-42-0.
- [80] Ramon A. Mata-Toledo and Pauline K. Cushman. *Schaum's Outline of Fundamentals of SQL Programming*. Schaum's Outline Series McGraw-Hill, 2000. ISBN 0-07-1359532-2.
- [81] Fraidoon Mazda. *Engineering Management*. Addison-Wesley Longman Limited, Harlow, Essex, England, first edition, 1998. ISBN 0-201-17798-6.
- [82] Microsoft Corporation. Microsoft Access Database, 2006. URL <http://office.microsoft.com/en-gb/access/default.aspx>.
- [83] Microsoft Corporation. Microsoft Dynamics, 2006. URL <http://www.microsoft.com/dynamics/default.aspx>.

- [84] Minq Software AB. DbVisualizer, 2006. URL <http://www.minq.se/products/dbvis/>.
- [85] Eric J. Naiburg and Robert A. Maksimchuk. *UML for Database Design*. The Component Software Series. Addison–Wesley, 2001. ISBN 0–201–72163–5.
- [86] Nehmer, Robert A. and Robinson, Derek. An algebraic model for the representation of accounting systems. *Annals of Operations Research*, 71:179–198, 1997.
- [87] John M. Nicholas. *Project Management for Business and Engineering Principles and Practice*. Elsevier Butterworth Heinemann, 2004. ISBN 0-7506-7824-0.
- [88] National Institute of Standards and Technology (NIST). Digital media formats, 2006. URL <http://www.itl.nist.gov/div895/formats.html>.
- [89] Open University Mathematics Foundation Course Team. *Relations*, volume Correspondence text Unit 19 of *Mathematics Foundation Course*. The Open University Press, 1971. ISBN 335 01018 0.
- [90] Oracle Corporation. Oracle’s peoplesoft enterprise applications, 2006. URL <http://www.oracle.com/applications/peoplesoft-enterprise.html>.
- [91] P. J. Pahl. Information Technology for Civil Engineering. In *Seminar on Engineering in Distributed Environments, Stellenbosch*. Department of Civil Engineering, University of Stellenbosch, October 2002.
- [92] P. J. Pahl and R. Damrath. *Mathematical Foundations of Computational Engineering*. Springer, 2001. ISBN 3–540–67995–2.
- [93] J. Paredaens, P. De Bra, M. Gyssens, and D. Van Gucht. *The Structure of the Relational Database Model*. Springer-Verlag, 1989. ISBN 0-387-13714-9.
- [94] O’Neill Patrick. *Database Principles, Programming, Performance*. Morgan Kaufmann Publishers, 1994. ISBN 1-55860-219-4.
- [95] Ronald R. Plew and Ryan K. Stephens. *SAMS Teach Yourself SQL in 24 Hours*. SAMS Publishing, third edition, 2003. ISBN 0–672–32442–3.
- [96] PostgreSQL Global Development Group. PostgreSQL open source relational database system, 2006. URL <http://www.postgresql.org>.
- [97] Philip J. Pratt. *A Guide to SQL*. Thompson Information / Publishing Group, 1991. ISBN 0–87835–669–X.
- [98] Purdue University - OnePurdue Project. OnePurdue Initiative, 2006. URL <http://www.purdue.edu/onepurdue/>.
- [99] B. H. Puttergill. Computers as a Management Tool in a Consulting Engineering Practice. In *Annual Conference on Computers in Civil Engineering – Computers 86*, April 1986. ISBN 0-7988-3692-2. ISBN 0-7988-3692-2.
- [100] Red Hat Inc. Red Hat Linux - The Fedora Project, 2006. URL <http://www.redhat.com/>.
- [101] Martin H. Rennhackkamp. *Database Application Development Analysts’ Manual*. The Data Base Approach Consultancy, The Database Consultancy cc. P. O. Box 5165, HELDERBERG 7135 Somerset West, 1990. ISBN 0-620-14913-2.
- [102] Michel Robert. *Strategy Pure & Simple*. McGraw–Hill, first edition, 1993. ISBN 0–07–053131–5.
- [103] James A. Robertson. An integrated, interactive professional practice management and accounting information system. In *11th Annual Conference on Computers in Civil Engineering – Information Systems in Civil Engineering*, May 1989.
- [104] James A. Robertson. A structural business model with the objective of improving profitability. *The Civil Engineer in South Africa*, 32(10):393–397, Oct 1990. ISSN 0009-7845.

- [105] Carla Rooseboom. The Management of Infrastructure and Buildings, focusing on the Management of Office Buildings. Project report presented in partial fulfilment of the requirements for the degree of Bachelors in Civil Engineering at the University of Stellenbosch S10/1997, University of Stellenbosch, October 1997.
- [106] Walldorf Germany SAP Aktiengesellschaft. SAP Business Software Systems, 2006. URL <http://www.sap.com>.
- [107] Norman M. Scarborough and Thomas W. Zimmerer. *Effective Small Business Management An Entrepreneurial Approach*. Prentice Hall, sixth edition, 2000. ISBN 0-13-080708-7.
- [108] A.-W. Scheer. *ARIS Business Process Modelling*. Springer, third edition, 1999. ISBN 3-540-65835-1.
- [109] Christian Schenk. The MiKTeX project page, 2006. URL <http://www.miktex.org/>.
- [110] Pieter Smit. Formal Business Model And Analysis Of Management System For Consulting Engineers In Professional Practice. Project report presented in partial fulfilment of the requirements for the degree of Bachelors in Civil Engineering at the University of Stellenbosch S9/2003, University of Stellenbosch, December 2003.
- [111] Softline Systems. Pastell payroll, 2006. URL <http://www.pastel.co.za/payroll/index.asp>.
- [112] Softline VIP. VIP Payroll, 2006. URL <http://www.vippayroll.co.za/>.
- [113] South African Association of Consulting Engineers. Standardised accounting system, 1993.
- [114] SSH Communications Security. SSH Secure Shell, 2006. URL <http://www.ssh.com/>.
- [115] S.K. Stanczyk. *Theory and Practice of Relational Databases*. Pitman, 1990. ISBN 0-273-03049-3.
- [116] J A v B Strasheim. Department of Water Affairs and Forestry (DWAF) Community Water Supply and Sanitation (CWSS) business process and workflow review. Technical Report ACTIONiT.10.20.50. Community water supply and sanitation process workflow report, Action IT Innovation Fund Project, February 2003.
- [117] J. A. v B. Strasheim. Facilities Management: A Civil Engineering Perspective. In *Eighteenth Annual Symposium on Information Technology in Civil Engineering – IT for Africa*, Oct 1996. ISBN 0-620-20657-8.
- [118] J. A. v. B. Strasheim, J. D. Krige, and J. H. Greyling. Die Welkom Waterplan – ’n Hulpmiddel vir Bestuur. In *Transaksies van die Instituut van Munisipale Ingenieurs*, 1990.
- [119] The Eclipse Foundation. Eclipse - an open development platform, 2006. URL <http://www.eclipse.org/>.
- [120] The Mathworks Inc. MATLAB, 2006. URL <http://www.mathworks.com/>.
- [121] The PostgreSQL Global Development Group. *PostgreSQL 7.4.2 Documentation*. The PostgreSQL Global Development Group, 2004.
- [122] J.G. Van der Merwe, R.B. Appleton, P.A. Delport, R.W. Furney, D.P. Mahoney, and M. Koen. *South African Corporate Business Administration*. Juta & Co. Ltd., 1995-2006. ISBN 0 7021 3326 4.
- [123] Rias J. Van Wyk. Panoramic Scanning and the Technological Environment. *Technovation*, 2: 101-120, 1984.
- [124] Ludwig Von Bertalanffy. General Systems Theory, 1968. URL <http://www.panarchy.org/vonbertalanffy/systems.1968.html>.
- [125] A. P. C. Warner. Implementation of a marketing strategy for GFJ Inc. Technical report, GFJ Inc., April 1989.
- [126] Wikipedia. Wikipedia, the free online encyclopedia, 2006. URL en.wikipedia.org.

- [127] Warren J. Wittreich. How to Buy / Sell Professional Services. *Harvard Business Review*, March–April 1966.
- [128] WSP Group. WSP Consulting, Environmental, Facilities Management, Projects and Energy Management, 2006. URL <http://www.wspgroup.co.za/>.
- [129] Laurie Young. *Marketing the Professional Services Firm*. John Wiley & Sons, Ltd, 2005. ISBN 0-470-01173-4.
- [130] yWorks - The Diagramming Company. yed - *JavaTM* Graph Editor, 2006. URL <http://www.yworks.com>.