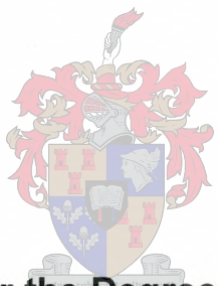
Structural analysis in a distributed collaboratory

by

Gert Cornelis van Rooyen



Dissertation presented for the Degree of Doctor of Philosophy
at the University of Stellenbosch

Internal promotor:	Prof. P.E. Dunaiski
External co-promotor:	Prof. Dr. Dr. h.c. mult. P.J. Pahl (Technische Universität Berlin)

25 November 2002

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.

Date: 25 November 2002

Synopsis

Structural analysis is examined in order to identify its essential information requirements, its fundamental tasks, and the essential functionalities that applications which support it should provide. The special characteristics of the information content of structural analysis and the algorithms that operate on it are looked into and exploited to devise data structures and utilities that provide proper support of the analysis task within a local environment, while presenting the opportunity to be extended to the context of a distributed network-based collaboratory as well. Aspects regarding the distribution of analysis parameters and methods are analysed and alternatives are evaluated. The extentions required to adapt the local data structures and utilities for use in a distributed communication network are developed and implemented in pilot form. Examples of collaborative analysis are shown, and an evaluation of the overhead involved in distributed work is performed.

Opsomming

'n Ondersoek van die struktuuranalise-taak word uitgevoer waarin die kern-inligtingsbehoeftes en fundamentele take daarvan, asook die vereisde funksionaliteit van toepassings wat dit ondersteun bepaal word. Die besondere eienskappe van struktuuranalise-inligting en die algoritmes wat daarop inwerk word ondersoek en benut om data strukture en metodes te ontwikkel wat die analise-taak goed ondersteun in 'n lokale omgewing, en wat terselfdertyd die moontlikheid bied om sodanig uitgebrei te word dat dit ook die taak in 'n verspreide samewerkingsgroepering kan ondersteun. Aspekte van die verspreiding van analise-parameters en metodes word ondersoek en alternatiewe oplossings word evalueer. Die uitbreidings wat nodig is om die datastrukture en metodes van die lokale omgewing aan te pas vir gebruik in verspreide kommunikasienetwerke word ontwikkel en in loodsvorm toegepas. Voorbeelde van samewerking-gebasseerde analise word getoon, en die oorhoofse koste verbonde aan analise in 'n verdeelde omgewing word evalueer.

Contents

1 INTRODUCTION AND RESEARCH FOCUS 1-1

1.1 Collaborative engineering and communication networks 1-1

1.1.1 Technology developments 1-2

1.1.2 Engineering application developments 1-5

1.2 Research focus 1-8

1.2.1 Guidelines for research on collaborative engineering environments 1-8

1.2.2 Environments for engineering software application development 1-9

1.2.3 Scope and research focus of the thesis 1-10

2 STRUCTURAL ANALYSIS OF BUILT FACILITIES 2-1

2.1 The structural analysis task 2-1

2.2 Numerical methods of structural analysis 2-3

2.2.1 Finite Difference method 2-3

2.2.2 Boundary Element method 2-3

2.2.3 Finite Element method 2-4

2.3 Finite Element analysis procedures 2-6

2.3.1 Linear analysis 2-6

2.3.2 Non-linear analysis 2-8

2.4 Structural building components 2-12

3 FINITE ELEMENT ANALYSIS OF THIN PLATES 3-1

3.1 Theory of thin plates 3-1

3.1.1 Idealisation 3-1

3.1.2 Kinematics 3-2

3.1.3 Material law 3-3

3.1.4 Stress resultants 3-3

3.1.5 Boundary conditions 3-5

3.1.6 Integral governing equations 3-7

3.2 Finite Element theory of thin plates 3-8

3.2.1 Finite geometry 3-9

3.2.2 Finite physical state 3-10

3.2.2.1 Description of the physical state 3-10

3.2.2.2 Internal interpolation 3-12

3.2.2.3 Interpolation of strain 3-12
3.2.2.4 Edge interpolation 3-13
3.2.3 Governing equations 3-14
3.2.3.1 Finite Element network 3-14
3.2.3.2 Contribution of the triangular elements 3-15
3.2.3.3 Contribution of the line elements 3-17
3.2.3.4 Algebraic governing equations 3-19
3.2.4 Results 3-19

4 OBJECT ORIENTED MODELS FOR STRUCTURAL ANALYSIS 4-1

4.1 Structuring of models and ordering of objects 4-1
4.1.1 Equivalence relation 4-2
4.1.2 Ordering relation 4-3
4.1.3 Mapping relation 4-5
4.1.4 Datastructure and ordering 4-6
4.1.4.1 Unsorted sequence 4-7
4.1.4.2 Sorted sequence 4-8
4.1.4.3 Sorting and searching 4-10
4.1.4.4 Datastructures and Java implementation 4-11
4.2 Scope of the implementation 4-12
4.3 Structure of an application 4-14
4.3.1 Objects and classes of an application 4-14
4.3.2 Relations between objects 4-15
4.3.3 Class App(lication) 4-16
4.3.4 Class AppObject 4-17
4.4 Implementation of the model components 4-18
4.5 Implementation of the model 4-21
4.5.1 Class Model 4-21
4.5.2 Class Analysis 4-22
4.5.3 Class Equation 4-24
4.5.3.1 Profile of the system matrix 4-24
4.5.3.2 Solution of the system equations 4-27
4.6 Class diagram 4-29

5 SCOPE OF MODELS FOR DISTRIBUTED COLLABORATORIES 5-1

5.1 General scope of the distributed analysis model. 5-1

5.2 Distribution of parameters 5-3

5.2.1 Analysis parameters of the project 5-3
5.2.2 Work pattern 5-4
5.2.3 Information units and granularity 5-5
5.2.4 Access control 5-6
5.2.5 Locking 5-6
5.2.6 Versioning 5-7
5.2.7 Persistent identification 5-7
5.2.8 Worksession modes 5-8
5.2.9 Parameter data volume 5-9
5.2.10 Persistent storage 5-10

5.3 Distribution of methods 5-12

5.3.1 Activation of references 5-12
5.3.2 Service methods 5-12
5.3.3 Methods at the workplaces 5-13
5.3.3.1 Modeling methods 5-14
5.3.3.2 Algorithmic methods 5-15
5.3.3.3 Interpretation methods 5-15
5.3.4 Distributed programming model 5-16

6 PARAMETERS 6-1

6.1 Persistent identification 6-1

6.2 Versioning 6-2

6.3 Access control 6-7

6.4 Consistency 6-8

6.5 Persistence 6-14

6.5.1 Storage location 6-14
6.5.2 Database technology 6-16
6.5.2.1 Serialization 6-16
6.5.2.2 Object oriented database 6-17
6.5.2.3 Relational database 6-18

7 METHODS 7-1

7.1 Computing platform 7-1

7.2 Activation of references 7-3

7.3 Collaboratory services 7-4

7.3.1 Project setting 7-4

7.3.2 Project analysis parameter database service (PAPD service) 7-5

7.3.2.1 Checking parameters into the project analysis parameter database 7-5

7.3.2.2 Checking a parameter out of the project analysis parameter database 7-8

7.3.3 Persistent identifier service 7-10

7.3.4 Consistency service 7-11

7.3.4.1 Workplace consistency service 7-12

7.3.4.2 Central consistency service 7-13

7.3.5 Versioning service 7-15

8 USER INTERFACE 8-1

8.1 Support of user programming 8-2

8.2 Explicit collaboration 8-3

8.2.1 Project analysis tasks 8-3

8.2.2 Project analysis parameters 8-3

8.2.3 Dynamic profile 8-6

8.3 Finite element analysis 8-7

8.3.1 Description of analysis parameters 8-7

8.3.2 Execution of the analysis algorithms 8-9

8.3.3 Interpretation 8-9

9 PILOT APPLICATION 9-1

9.1 Scope of the application 9-1

9.2 Structure of a distributed application 9-2

9.2.1 Objects, classes and services of a distributed application 9-2

9.2.2 Class App(lication) 9-3

9.2.3 Class AppObject 9-3

9.3 Implementation of the model components 9-4

9.4 Implementation of the model **9-5**
9.4.1 Class Model 9-5
9.4.2 Class Analysis 9-6
9.4.3 Class Equation 9-6
9.5 Implementation of the collaboratory services **9-8**
9.5.1 Project analysis parameter database service (PAPD service) 9-8
9.5.1.1 Project analysis parameter database (PAPD) 9-8
9.5.1.2 PAPD server 9-11
9.5.1.3 PAPD client 9-13
9.5.2 Persistent identifier service (PID service) 9-14
9.5.2.1 PID server 9-14
9.5.2.2 PID client 9-14
9.5.3 Consistency service 9-16
9.5.3.1 Workplace consistency service 9-16
9.5.3.2 Central consistency service 9-18
9.5.4 Versioning service 9-19
9.5.4.1 Versioning server 9-19
9.5.4.2 Versioning client 9-19

10 EXAMPLES AND EVALUATION **10-1**
10.1 Examples **10-1**
10.1.1 Example 1: Basic modeling, storage and retrieval. 10-2
10.1.2 Example 2: Using parameters in consistent mode 10-10
10.2 Evaluation **10-14**
10.2.1 Procedure 10-15
10.2.2 Results 10-16
10.2.2.1 Checking parameters into the PAPD 10-16
10.2.2.2 Checking parameters out of the PAPD 10-20
10.2.2.3 Versioning 10-23
10.2.2.4 Persistent identification 10-25
10.2.2.5 Propagation of changes to consistent parameters 10-27

11 CONCLUSION **11-1**

List of Figures

Figure 4-1:	Application class diagram	4-29
Figure 5-1:	Distributed structural analysis collaboratory	5-2
Figure 5-2:	Storage volume of analysis parameters	5-9
Figure 5-3:	Project server of an analysis collaboratory	5-16
Figure 6-1:	Parameter distribution	6-1
Figure 6-2:	Analysis parameter direct dependency graph	6-3
Figure 6-3:	Parameter access tag	6-8
Figure 6-4:	Equation attributes and parameter influences	6-9
Figure 6-5:	Parameter check-out procedure	6-11
Figure 6-6:	Consistent parameter modification	6-12
Figure 6-7:	Creating a new version of a consistent parameter	6-12
Figure 6-8:	Parameter check-in procedure	6-13
Figure 6-9:	Collaboratory architecture	6-14
Figure 6-10:	Architecture with OO database	6-17
Figure 6-11:	Impedance mismatch	6-19
Figure 6-12:	Architecture with relational database	6-19
Figure 6-13:	Three-tier architecture	6-20
Figure 7-1:	Object map in application address space	7-3
Figure 7-2:	Project analysis parameter database service	7-5
Figure 7-3:	Persistent identifier service	7-10
Figure 7-4:	Workspace and central consistency services	7-11
Figure 7-5:	Versioning service	7-15
Figure 9-1:	Model component parameters database tables	9-9
Figure 9-2:	Storage structure for models	9-10
Figure 9-3:	Classes comprising the PAPD server	9-11
Figure 9-4:	Classes comprising the consistency service	9-16
Figure 10-1:	Layout of example 1	10-2
Figure 10-2:	PAPD storage of model for task x1Geo	10-6
Figure 10-3:	PAPD storage of equation created in task x1L&A	10-9

List of Tables

Table 4-1:	Java data structures	4-11
Table 5-1:	WAN latency	5-8
Table 6-1:	Fundamental changes to analysis parameters	6-4
Table 7-1:	Fundamental attributes of analysis parameters	7-7
Table 9-1:	Database table names	9-8
Table 9-2:	Variables present in all database tables	9-9
Table 10-1:	Parameter check-in operations and data volumes	10-17
Table 10-2:	Flushing a packaged parameter	10-18
Table 10-3:	Parameter check-in times (particular configuration)	10-19
Table 10-4:	Parameter check-out operations and data volumes	10-20
Table 10-5:	Reading a packaged parameter	10-21
Table 10-6:	Parameter check-out times (particular configuration)	10-22
Table 10-7:	Versioning operations and data volumes	10-23
Table 10-8:	Version updating times (particular configuration)	10-24
Table 10-9:	Persistent identification operations and data volumes	10-25
Table 10-10:	Consistency operations and data volumes	10-27

Acronyms

AEC:	Architecture-Engineering-Construction
API:	Application Programming Interface
CORBA:	Common Object Request Broker Architecture
DFG:	Deutsche Forchungsgemeinschaft
FTP:	File Transfer Protocol
HTTP:	Hypertext Transfer Protocol
IAI:	International Alliance for Interoperability
IDL:	Interface Definition Language
IFC:	Industry Foundation Classes
IP:	Internet Protocol
ISO:	International Standards Organization
JDBC:	Java Database Connectivity
JRE:	Java Runtime Environment
JVM:	Java Virtual Machine
LAN:	Local Area Network
ODBC:	Open Database Connectivity
OMG:	Object Management Group
ORB:	Object Request Broker
PAPD:	Project Analysis Parameter Database
PID:	Persistent Identifier
RMI:	Remote Method Invocation
SID:	Selection Identifier
SMTP:	Simple Mail Transfer Protocol
SOAP:	Simple Object Access Protocol
SQL:	Structured Query Language
STEP:	Standard for the Exchange of Product model data
TCP:	Transmission Control Protocol
TELNET:	Network Terminal
UMTS:	Universal Mobile Telecommunications System
WAN:	Wide Area Network
XML:	Extensible Markup Language

1 Introduction and research focus

An extensive research effort, aimed at addressing the problems of designing a technology that can truly support distributed, collaborative Engineering, is under way. This is a worldwide phenomenon, and is supported by both the private sector and Governments through their respective local agencies, as well as international agencies. An example is the recent approval, by the Deutsche Forchungsgemeinschaft (DFG), of a Priority Program, "Vernetzt-kooperative Planungsprozesse im Konstruktiven Ingenieurbau" [Rüppel 1999], which addresses these problems in the field of structural engineering. The work described here represents a contribution to that effort.

The paragraphs below give a brief overview of relevant developments regarding collaborative engineering, communication networks, and the growing merging of the two under the concept of network-supported distributed, collaborative engineering. This is followed by a description of the research focus of the dissertation that indicates the originality and scientific content of the work.

1.1 Collaborative engineering and communication networks

In modern engineering projects it is normal for a number of professions, e.g. Architecture, Civil and Structural Engineering, Heating-Ventilation-Airconditioning Engineering, and Process Engineering, to be cooperatively involved in engineering projects, in a process that comprises four core tasks:

1. Planning transforms mental concepts into facilities with specified geometrical and physical properties, which are expected to satisfy the demands of usability, time, cost, quality and safety.
2. In the analysis phase, the behaviour of the planned facility is investigated to assure its conformance to technical, legal, economical, and social requirements. This phase includes the analysis of physical behaviour, the analysis of tendering-awarding-billing in the construction process, the analysis of financing and cash flow, and the analysis of the anticipated operation of the facility.
3. Structural design is the task in which the structural components and connections are selected which satisfy the assumptions of the analysis, as well as the requirements of the construction process, of the operation and of potential removal of the facility.
4. Construction leads from the planned to the real facility. It is a complicated process involving company management, site management, logistics, approval and checking, the building process, quantity surveying and financial management, scheduling and quality control.

The relations between the activities in the core tasks form a directed graph of high complexity, which varies from project to project, and contains numerous loops. The interaction be-

tween the persons and institutions involved in these activities should be as collaborative as possible. The process of interaction can be supported extensively with information- and communication technology. This technology provides both general support for administrative, technical and commercial tasks, as well as specific civil engineering software e.g. for draughting, analysis, planning, design and costing of facilities.

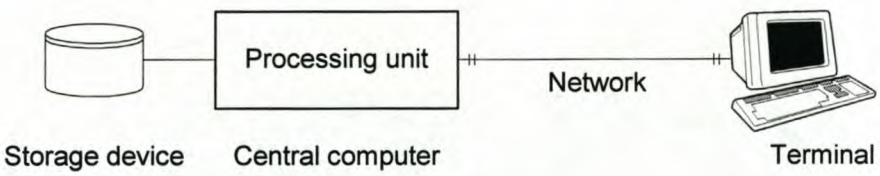
The worldwide growth of open communication networks, and their attractiveness to the professions as well as the business world, has changed the environment in which the core engineering tasks are performed. The persons involved in a project tend to be distributed over a larger number of geographic locations and tend to be affiliated with a larger number of institutions than before. Communication between them tends to change from close contact in an office and traditional forms of documentation such as contracts, reports and drawings, to communication that is based on computer networks. These networks connect many locations and users through a large variety of interactive forms of documentation, information and presentation. The trends described above offer new potentials for acceleration and improvement in the performance of the core engineering tasks. However, they also contain the risk of loss in quality of work, and of reduction in efficiency and accountability of engineers, as well as their capacity to maintain an overview of the problem at hand.

The trend towards distributed collaboration in engineering, mentioned above, has its roots in a synergy between computer and communication technology developments, and engineering application developments.

1.1.1 Technology developments:

The rate of development of digital computers and networks is well documented [Rausch 1991 Table 1.1, Minoli 1997 Table 1.2]. From the vantage point of their influence on distributed, collaborative engineering, three milestones in the development of communication technology are listed below:

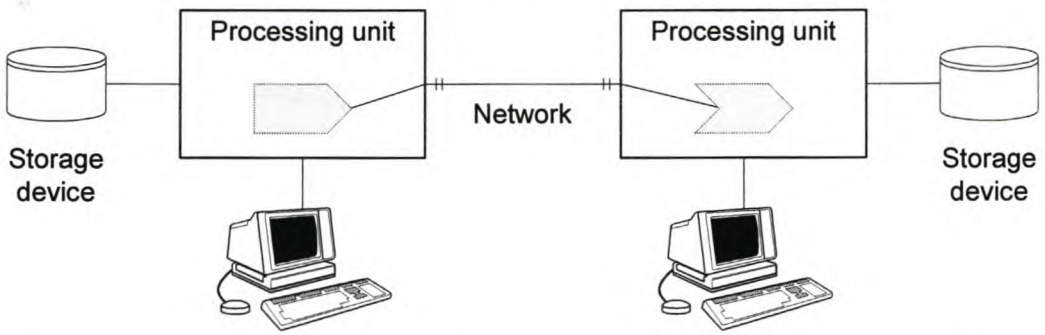
1. **Computer-to-terminal communication:**



In the period from the early 1960s to the 1980s, computer aided engineering was primarily performed on mainframe computers. With increasing use of mini computers such as the DEC/VAX and Prime systems, significant computer power was made available to the technical offices. It became a problem to make this power accessible to the individual engineers. At this time simple access networks were installed which connected a distributed set of terminals to a central computer [Harms 1993]. The terminals were usually distributed within a single institution, and were used solely as devices for input and output of

data. All processing took place on the central computer and the operational goal was simply to process as many programmes as possible. The level of software and network development did not allow much user-friendliness, e.g. interactive communication with a user. The configuration, as described, was not conducive to effective or wide-spread collaboration.

2. **Application-to-application communication:**

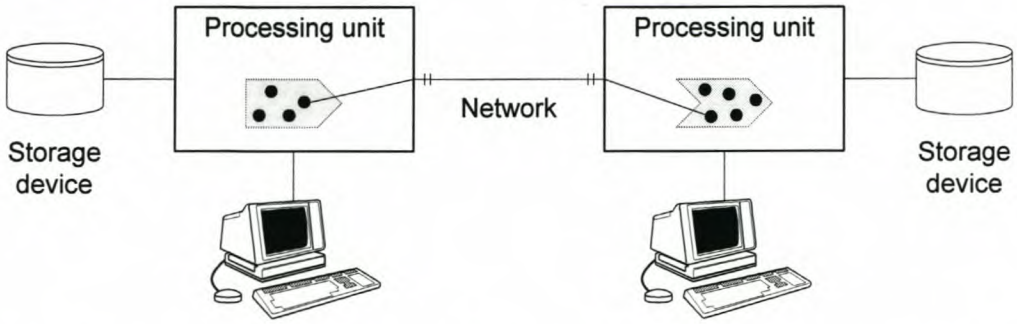


During the 1970s remarkable progress was made with the standardisation of network operating environments, as well as the hardware technologies which these would control. By 1983 the Transmission Control Protocol (TCP) and Internet Protocol (IP), forming the core of the Internet protocol suite, provided the standard operating environment of systems connected in local area networks (LAN), and of systems which established an inter-network connection called the Internet [Minoli 1997]. Based on the TCP/IP suite, application layer protocols were developed which provided network services like TELNET (Network Terminal), FTP (File Transfer Protocol) and SMTP (Simple Mail Transfer Protocol). The configuration and services at this level of technical development essentially provided application-to-application communication on a client-server basis, in which the following was possible:

- local execution of interactive applications,
- batch execution of remote applications (due to low network capacity)
- input and output at the user surfaces of the terminal computers, and
- transfer of files in which the structure of files could be maintained.

Under these circumstances distributed collaboration could be implemented as a sequential cooperative process. An example dealing with computer aided draughting is presented in [Ilieva 1997]. In this work it is important to note that care was taken to minimize network traffic to the extent that only the differences to previous version drawing files were transmitted, instead of the complete updated file. At the time this might have seemed overly conservative, or even pessimistic regarding growth in network capacity. However, later investigations showed that, although network capacities have been growing, internet communication would remain relatively slow and unreliable [Chan 1998, WELD].

3. **Object-to-object communication:**



From the point of view of developing distributed applications, a much higher level of functionality can be achieved if communication can be established at the object level between distributed applications, and is not limited to communication between applications. The programming language Java [Java] provides such a mechanism through its Remote Method Invocation (RMI). The limitations of RMI are that the location of the remote object has to be known, and that the same programming language (Java) has to be used at the local and the remote sides of the application [Boger 1999]. These limitations were overcome in the Object Management Group's (OMG) specification of the Common Object Request Broker Architecture (CORBA) [OMG]. CORBA technology makes communication at the object level possible between component-applications, independent of the objects' location or the programming languages of the component-applications. Current CORBA-compliant implementations do not yet provide for the transparent migration (transport) of objects, a utility which is envisaged for future releases. In spite of this shortcoming, the current Object Request Brokers (ORB), conforming to the CORBA standard, enable developers to implement software in which the data flow and the work flow in the distributed applications can be controlled. In principle, therefore, current communication technology meets the technical requirements of applications which could support sequential, parallel or reciprocal cooperation in a distributed environment. However, the complicated collaborative work process with its heterogeneous information base and special management requirements can only be adequately modeled at a higher level technology. The available communication technologies on their own do not solve the problems of distributed collaboration, but they will form the basis of such a higher level technology, and as such they do open up new possibilities [Eastman 1997]. Current research efforts are aimed at exploiting these possibilities

1.1.2 Engineering application developments:

The computer's capabilities have been utilised by engineers very early in its history. The focus of how computers were used, however, was influenced by the changing capabilities of the computer operating environment and external influences on the engineering professions. A number of phases in civil engineering computer usage patterns are listed below.

1. **Computation:** The computer's computational capabilities were immediately recognised and utilised for the execution of algorithms with high demands on computation. An example is the Finite Element Method, the development of which in the late 1950s and early 1960s was stimulated by the availability of a tool which could perform the necessary computations [Heubner 1982 p. 11]. Although computational capability may no longer be at the centre of computer usage, its continuing significance for practical engineering work should not be underestimated.
2. **Information storage:** In engineering the principal information carrier has always been drawings. As they became available, computer graphics capabilities were utilised in draughting applications. Work in this area has been summarised under the term Computer Aided Design (CAD), which is a misnomer since the majority of CAD packages are in fact draughting systems. The CAD applications made modifications, so typical throughout the development of engineering projects, much easier. The incorporation of macros and templates further facilitated and simplified the work. The computer was thus utilised for the storage of information. In the early 1980s the Personal Computer (PC) and the conceptually similar but more powerful workstations were developed, which could provide most of the processing capacity required in engineering applications, at a cost which even smaller engineering offices could afford. These came to be widely used for both technical and administrative tasks. Since the PC made storage capacity available to a large section of the profession, the tendency to use the computer as a storage medium for both text and drawings increased significantly.
3. **Information exchange:** The computer network was originally seen solely as a mechanism for connecting components. However, the physical configuration of powerful distributed processing capacity, connected by a network which provided services like TELNET, FTP and SMTP, changed that perception fundamentally. It became clear that the network formed the core of a distributed system [Harms 1993]. Within the distributed system, the exchange of information generated during the execution of tasks became important. Inputs and results of applications were stored in files and made available to other applications through transfer of the files. This speeded up the engineering process and diminished the error-prone human interaction during the data-entering phases. Early attempts at information exchange based on file transfer were burdened by the fact that, due to the procedural programming languages used in application development, it was difficult to separate the semantics of information from the algorithms used in the applications. As a result, the effort of transporting data between applications was prohibitive. In

addition to the network communication effort, the effort required to transform data structures from the conventions of one application system to another proved very significant. The semantics of information were separated from the algorithms by the employment of abstract datatypes, later followed by fully object-orientated applications. This prepared the way for meaningful exchange of information.

4. **Standardisation:** It was recognised that the building industry (also called the Architecture-Engineering-Construction or AEC industries) is an intensive user of information, and that big savings could potentially be made if the information flow during project development could be facilitated by standardisation. The ISO 10303 "Standard for the Exchange of Product model data" (STEP) [ISO10303] may be considered the most significant standardization effort. STEP supports the development of the Building Construction Core Model and Application Protocols for the building industry. However, due to the complex nature of products and processes in AEC projects, standardizing AEC information, and finding ways to deal with the processes in a distributed computer environment, is a difficult task [Fischer 1996]. Researchers tried to address these problems in a number of projects, summarized in [Froese 1996]. In spite of these efforts, and in spite of the streamlining of information exchange offered by standards based solutions, their implementation in commercially developed engineering software have been disappointingly slow. Even for the widely used CAD applications no generally accepted standard for the data- and file structure exists. A contributing factor is the complicated nature of some of the proposed solutions. [Rosenman 1996]. Furthermore, the aim of standards such as STEP is the exchange of product data in a highly systematised and generalised form. It is not the aim to provide an implementation model for efficient operations on these product data, as required in a runtime environment for collaborative design and engineering. It is generally agreed that the design and engineering phases require additional models and user interfaces for local operations. A more recent development is the International Alliance for Interoperability's (IAI) effort to define Industry Foundation Classes (IFC) [IAI], i.e. a semantic object model for the building industry. The IAI is an alliance of companies with an interest in the building industry which is creating and promoting the IFC specifications as a basis for information sharing through the life cycle of a building project. Even though the IFC represent something like an "implementable STEP", general acceptance, and implementation of this specification is also still a long way off. Cooperation between STEP and the IAI, and coordinating their efforts, hold promise for increasing the tempo of standards development, as well as improving their chances of general acceptance. [STEP-IAI1997].
5. **Collaboration:** The initial efforts aimed at supporting collaborative engineering in a computer network concentrated mostly on models for information exchange. Issues which are important in collaborative environments, like ownership, rights and responsibility, versioning of information, schema evolution, recording the intent behind decisions and notification and propagation of changes, was considered in [Rezgui 1996/7/8, Brown

1996]. However, supporting distributed collaboration to a degree higher than what can be accomplished by information exchange have become important. At the root of it lies a fundamental, structural change in civil engineering [Pahl 1999]. In current practice, an overall view is taken regarding planning, design, construction and usage of the built environment and natural systems, including the impact of social, financial and economic aspects. Competition takes place on a global scale, and engineers must deal with a more complex environment, a higher state of the art, and a heterogenous composition of professions represented in building projects. As a result, competitiveness in civil engineering, in earlier times determined by the exploitation of better materials, enhanced methods of construction and innovative design of building systems, is now equally dependend on effective organization and management, which in turn are critically dependend on information and communication. The information flows and communication requirements in distributed, heterogenous project teams have to be supported. Application of the potentials offered by the major technology developments which have taken place in communication networks have become critical for survival in the civil engineering industry.

Current research into developing a technology which can support distributed collaborative engineering in a communication network represents, in a sense, a second generation effort brought about by the developments and pressures described above. There is an expectation that success can be achieved. Bretschneider [Bretschneider 1998], e.g., showed that, in principle, a neutral process model for collaborative engineering work could be realised, and tested it in the design of steel framed structures.

The degree of success which will be achieved with a collaboration-supporting technology, however, will depend on the extent to which this technology makes it possible for the persons involved in a project to work efficiently. This requires not only the exchange of information, but synchronous communication between the different parties. Furthermore, the degree of dependence on the developers of engineering software should be minimised. It should be accepted that the applications used by members of project development teams will continue to be developed independently, even though there is the demand that they be applied coherently. This can be accomplished by actively involving expert users of applications in the integration process, as described in [Schneider 1999].

In the structural engineering field these issues will, over a six year period, be researched in depth in the DFG priority program [DFG SPP 1103]. Successful implementation of the concepts and technology flowing from research programmes such as this will make a significant difference to the competitiveness and development of the civil engineering profession, especially in the long term.

1.2 Research focus

1.2.1 Guidelines for research on collaborative engineering environments

Process and product: The core engineering tasks described in section 1.1 collectively comprise a process which transforms mental concepts into real facilities. A facility is in generic terms a product which is developed during the engineering process. The process comprises a number of sub-processes, most of which are supported by software tools. These tools operate on information objects which collectively constitute the software models of the product. A software model is a structured set of information objects and relations which serves to represent a specific view of the product. The product model is a structured set of information objects and relations needed to describe the product. In general, the software tools used in current engineering practice do not implement a common, semantically consistent and comprehensive product model as a basis for the models they respectively support.

Requirements: A whole range of factors are important to distributed collaboration, e.g. economical, social, legal, ethical, institutional and technological issues. From a technological viewpoint the aim of building online information environments for collaborative engineering is to allow team members to remain in their physical workplace without being constrained by its limitations. For a virtual environment to be effective, it must blend unobtrusively with the physical environment of the user [Rosenberg 2000]. This implies that, in the design and choice of collaboration media, the environment has to meet the technical requirements of distributed collaboration, while accounting for human issues as well [Oberquelle 1991].

- ***Technical requirements:*** Within the virtual workspace created by a distributed collaborative technology, team members have to execute their respective tasks. They do this using software tools which operate on information objects. A variety of software tools may be used. Collaboration means that the team members work together to achieve the same goal, and that their decisions and actions are based on their mutual understanding of the goal and the information they share. The shared information is contained in the product model. During the engineering process the product model is constantly manipulated by team members to reflect desired states. This collaborative manipulation of the product model requires a fusion of the engineering process, supported by the tools operating on the product model, and the product, represented by the product model itself. The necessary sharing of information which has to occur during this merger implies that the semantics of information contained in the respective sub-models have to be compatible. The proposed [DFG SPP 1103] programme, mentioned in section 1.1, accounts for the required fusion of process and product by specifying that research should be conducted with a view to the following:

- From the project point of view, the relevant work-processes are to be systematically analysed in order to divide these processes into generic, cooperative sub-processes with compatible information flows and interaction.
- From the point of view of engineering models and software, concepts for integration and management of information are to be developed and tested.

- ***Human issues:*** Human issues vary from a simple predisposition to use or avoid certain media, to concerns about different mentalities and cultures of the collaborating engineers. These issues are researched, for example at Stanford University's Project Based Learning Laboratory [PBL Lab] and Center for Integrated Facility Engineering [CIFE]. The [DFG SPP 1103] research programme also recognised, at least to some extent, the importance of human issues, and specifies that:
  - From the point of view of the people involved in the project, the communication requirements and the technical requirements of the different types of communication, in a distributed system, are to be determined.

1.2.2 Environments for engineering software application development

The development of software for specialised civil engineering applications can be treated in two significantly different environments, namely the general software platform, and the civil engineering (specialised) software platform.

General software platform: The information structure and communication methods of the general software platform are based on the assumption that the contents of the information and the algorithms of the problem solving procedures are unknown. Only general features of communication, such as establishing connections between partners, aggregation of information in files, packaging in data transfer, message services, etc., are provided. It is to be expected that this approach has the advantage of generality and flexibility, but does not make use of the characteristics of the professional area for which the computer support is developed. It can therefore not be expected that the general software platform provides optimal support for engineering tasks.

Civil engineering platform: From the viewpoint of civil engineering applications, the aim is to provide computer support for the engineering process. Particular processes are supported through the provision of specialised tools which operate on appropriately structured information. The usefulness and efficiency of these tools become the central issue. Characteristics of the special processes and distinct information requirements can be used to develop and employ particular algorithms, data structures and communication methods which efficiently support the processes in a manner which the general platform, without knowledge of the civil engineering processes and information objects, cannot achieve.

1.2.3 Scope and research focus of the thesis

The scope of a general investigation into information structures and processing methods for collaborative civil engineering in distributed environments is too broad for complete in-depth treatment in a single dissertation. In order to permit a detailed and systematic analysis, the investigation is focussed on a few well defined issues of particular importance.

Scope: The scope of the thesis is narrowed by focussing on one of the prime tasks of civil engineering, namely structural analysis. This task owes its importance to the fact that it usually plays a significant, and at some stages a crucial role in all of the phases of civil engineering projects. Referring to the core tasks of the civil engineering process, i.e Planning, Analysis, Design and Construction, described in section 1.1, we find that structural analyses usually occur in all of them:

1. During the planning phase the search for an economical solution which meets the criteria, the evaluation of alternative proposals involves preliminary structural analyses. During this phase a large amount of information is generated and interaction between project participants is intensive.
2. Once a specific proposal has been accepted a detailed structural analysis is required as part of the analysis of the physical behaviour of the facility. The structural-physical behaviour of a facility has to be adequate as measured against serviceability limit states.
3. Structural analysis delivers the basic information, e.g. stresses, strains and the resultants of these, which structural design needs in its task of selecting suitable components and connections. Acceptable design solutions are not easy to find, and the analysis $\leftrightarrow$ design loop usually involves a number of iterations. Furthermore, since the requirements of the construction process, of the operation and of potential removal of the facility have to be considered, this process has to be repeated for each of them.
4. During construction it is inevitable that last-minute changes and special construction techniques will require additional structural analyses in order to ensure useability, quality and safety.

Given the important role it plays in civil engineering endeavours, it is not surprising that structural analsysis has been studied scientifically for centuries. As a result it has a very good mathematical foundation, because of which structural analysis has benefited greatly from the development of computers. The extensive degree and advanced nature of software applications supporting structural analysis is proof of this [Abaqus, Adina, Nastran, mb, Nemetcheck].

General aim: With reference to the statements above, the investigation will concentrate on the processes, the models and the communication requirements of structural analysis with the general aim of supporting this task in a distributed collaboratory. The relative merits of exploiting the civil engineering software platform in the design of processing tools and data structures, as opposed to relying only on the general software platform, will be evaluated.

Scope limitation: Restricting the scope of the thesis to structural analysis still leaves a very broad field of investigation. Structural analysis is a complex task, involving a large variety of possible analysis types and solution approaches. The range and complexity of structural analysis will be described in relative detail in a following section. No effort will be made in this thesis to catalogue the information and methods to cover all of structural analysis. Instead the essential features of structural analysis in a distributed collaboratory will be identified, alternative solutions to the problems will be analysed, and preferred methods of solution will be proposed and illustrated in a pilot study.

Procedure: Structural analysis of built facilities in a distributed collaboratory demands a specific structure of information and communication. This structure will be investigated in depth in this thesis. In particular, structural analysis in a distributed environment will be analysed to identify

- ***Essential information requirements:*** Exchanging information is one of the cornerstones of collaboration. In a software environment information is generated and manipulated by software tools. A large number of commercial software tools support the structural analysis task. These tools are developed in separate environments and do not derive their particular information objects and analysis models from a common, holistic, semantically consistent product model. Consequently, in order to exchange information within a collaboratory, mapping of information components (object attributes) is required. If the common component of the model, i.e. common information which engineers working in the collaboratory need to gain contact with, is well defined, the mapping can be implemented as extentions to the software tools. This creates a basis for systematic information exchange and paves the way for the required fusion of the process and product. The aim here is not to develop a comprehensive analysis model. However, it is expected that the investigation will show that a large part of the common component of the analysis model can be defined. We call this component the essential information requirements of structural analysis. The aim is to examine the essential information requirements to identify special relations and properties (unique to structural analysis information and processes) with the intent of exploiting these special characteristics in the structuring of information in the distributed environment. In the same vein special restraints on the management of structural analysis information have to be identified and accounted for in the information structure.
- ***Fundamental tasks:*** The aim of this investigation is to find the abstract tasks structural engineers do in a collaboratory. Structural analysis is a highly technical, but also a creative task. For this reason the distributed design environment should not force engineers to follow rigid procedures. To allow the required freedom, but still provide the support needed to work efficiently, the virtual workspace has to support the abstract tasks. Furthermore, the interdependence of tasks in a collaboratory determines to what extent these can be excecuted parrallel in time, as well as the notification requirements amongst interdependent tasks.

- ***Essential functionality:*** The fundamental tasks described above have to be supported by capable software tools. In spite of the large number of available tools supporting structural engineering tasks, very few of these are equipped to operate in a distributed environment, even less actively support collaboration in any form. In general, existing systems operate in a monolithic configuration, manipulating some part of the product model in isolation of other tools and product model components. The aim of this part of the investigation is to identify the essential functionalities needed to support the fundamental structural analysis tasks. Algorithms accounting for these functionalities have to be investigated as well, since they depend on, and influence, the data structure.

In summary, the investigations described above will concentrate on the technical requirements of distributed structural analysis. The focus will be on exploiting the special characteristics of its essential information requirements in order to devise utilities and data structures which encourage distributed analysis through effective support of its fundamental tasks. A systematic comparison of alternate solutions will indicate preferential directions of future development. The relative merits of civil engineering platform solutions and general software platform solutions for data structures, algorithms and user interfaces will be evaluated in terms of compliance with the functional requirements of distributed analysis and the criteria stated below.

Criteria: Criteria for successful collaboration in distributed environments are numerous and vary with respect to the specific problem area which is under consideration. Borghoff [Borghoff 1998, Chapter 4] lists a number of general criteria. More specialised criteria, relevant to a neutral cooperation model for distributed engineering, are listed in [Bretschneider 1998]. The criteria stated below were selected due to their bearing on what can be achieved in terms of the research focus of the thesis. At the same time, however, these criteria are important to the management of information and provision of utilities in any distributed collaboratory.

Criteria pertaining to the management of information:

- ***Standalone capability:*** An environment involving a number of computers and an Internet-based communication system carries a high risk of failure in any one of its components. Such an occurrence should not incapacitate the complete environment. Working at unaffected stations should be possible in isolation. This implies that information which is essential to the tasks performed at a certain location, have to be stored at that location, but managed in the distributed environment.
- ***Consistency:*** Members of the project team make use of information available to them at a certain point in time. If the information base is not consistent, the decisions and actions they take are based on outdated information. Such situations have to be avoided since they can have a negative influence on the safety and useability of the structures, on the economics of the project and on the time schedule. Alternative methods of accounting for

consistency [Borghoff 1998] have to be evaluated in terms of the impact they have on team members working in the collaborative environment.

- ***Flexibility:*** The predefinition of models and other structured sets lend a useful structure to the information base. However, as the project evolves, models and sets may have to change, or be removed, and new ones formed. As a result the manipulation of models, objects, sets of objects and set-forming relations need to be flexible.
- ***Media-suitability:*** Recent advances in computer networks have connected the computer to a multitude of conventional and novel media. These include displays, printer, scanner and audio devices. Advanced research also considers various tools for virtual reality, e.g. creating a three-dimensional stereoscopic environment for conceptual design [Kwaw 2000], or utilising position sensors, orientation sensors and tactile interface devices to create an overlay between real-world and virtual objects in an augmented reality environment [Kensek 2000]. There is much speculation about the future shape, size and material of computers. In view of this broad spectrum, it is essential to concentrate the research on those media which promise not only to be interesting from the point of view of information technology in general, but useful for the special requirements of civil engineering practice. Much hope was initially attached to new forms of visual interaction on displays, e.g. through video conferences supported by the display of documents [Molkenthin 1998]. The success of these methods has been less pronounced than expected. This may be attributed to the fact that a significant part of a project's information, particularly the interdependence between different pieces of information, is person-based and is conventionally exchanged by speaking. Apart from information exchange, another important goal of the speech act is the requesting and fullfilling of commitments between the conversation participants [Turk 2000]. Both of these acts, i.e. requesting commitments and fullfilling them, are essential in a collaboratory. Consequently it is not surprising that experience in design offices has indicated that verbal communication amongst team members is of cardinal importance in the collaboration process. Scientific studies by various authors [Fruchter 2000, Al-Qawasmi 2000, Rosenberg 2000] support this observation. In distributed environments, however, the danger of ambiguity about what exactly is being discussed is real. Supporting participants in a conversation to simultaneously look at the same drawing, and to point at objects in the drawing while conversing about issues of importance, will greatly enhance and encourage distributed collaboration. In the near future users will make use of mobile devices, e.g. handheld computers connected to Universal Mobile Telecommunications System (UMTS) networks. Such devices present special problems as a result of their size and the speed and reliability of mobile communication. Complete models comprising of large data volumes cannot be transmitted or viewed and processed. The information structure has to allow for the selection of small units of information according to different engineering semantics based criteria.
- ***Navigability:*** In a collaboratory, team members are constantly looking for and exchanging information. It is therefore essential that members are able to find information as eas-

ily and quickly as possible. A known problem of solutions attempting to address this navigation problem is that pre-programmed conventions regarding search criteria may not be suitable for users of such a system at all. A more flexible navigation system is required.

- ***Updating the common information base at user-specified points in time:*** The problem of maintaining a consistent information base is a key issue in distributed environments [Pahl 2000]. Suitable change propagation algorithms are still under development, and continuous change propagation may not be practical. Furthermore, the creative nature of engineering solutions implies that some alternatives which team members choose to investigate may not be of common interest, and should not persist in the project databases. Enabling users to control the updating time of the common information base provide the opportunity to solve compatibility problems between versions as well.
- ***Updating the common information base in user-specifed areas:*** It is expected that the survey of the structural analysis task will indicate that the effect of certain changes (which may be made frequently) may be localised in the information database. Also, users may want to retain some of the modifications made during a worksession, while discarding others. Enabling users to specify an area of the information base which requires updating enhances user control and flexibility of data management.
- ***History of decisionmaking:*** The development of a project critically depends on decisions taken as events unfold. A decision taken at a certain point in time should take relevant earlier decisions into account. A reconstructible history of decisions affecting the project, which also captures the intent behind the decisions, will enhance the project members' understanding of the reasons behind changes. Consequently, it reduces misunderstandings, facilitate backtracking, and provide an overview of the project history.

Criteria pertaining to the performance of distributed software utilities:

- ***Reliability and safety:*** Typical sessions in an engineering environment are not short. A session may last hours or even days, and consist of hundreds of transactions. Loss of work due to system and network failures should not occur. An Undo/Redo facility is also necessary to minimise the effect of user mistakes.
- ***Maintaining a feeling of collaboration:*** The interdependence of various fundamental tasks is important. It is necessary that engineers should be informed about relevant acts by other team members, while at the same time realising that their own actions may have an impact on what others are doing.

It is expected that significant advantages can be gained by exploiting the potentials of the civil engineering platform in comparison to black box solutions in general platforms. It is furthermore expected that the results of the investigations will indicate a need for similar investigations in other civil engineering fields of speciality, and that proposed solution techniques may be applicable to other specialised applications as well.

2 Structural analysis of built facilities

The structure of information and communication needed for structural analysis in a distributed environment is under investigation. This requires an examination of the structural analysis task, with the aim of identifying essential information requirements[1], fundamental tasks[2] and key functionalities[3] required to execute those tasks in a distributed collaboratory.

In this chapter concise descriptions of numerical techniques of structural analysis are given, emphasising Finite Element procedures. This is followed by a brief account of common structural building components. This chapter highlights the richness of techniques and data content of structural analysis, whose wide scope and complexity justify the necessity of limiting the detailed examination and pilot implementation in the thesis to a typical application. Finite element analysis of thin plates is chosen for this purpose. As a background to that, the Kirchoff theory of thin plates and a finite element theory for the solution of its governing equations are summarised in the next chapter. These provide the particulars of the datatypes, and the algorithms operating on them, described in chapter 4.

2.1 The structural analysis task

Structural analysis: Structural analysis is the process whereby the physical behaviour of built facilities is predicted. The behaviour (i.e. response) of a facility is a measure of how it reacts to given external influences. The laws of physics describe the actual relation between influences on an object and its physical response. The brief of structural analysis is to mathematically describe the relation between influences and response for structural components, and to find a solution to the equations for the problem at hand. The physical laws governing structural mechanical behaviour, e.g. the laws of motion and laws describing the behaviour of engineering materials, provide the basis of all analysis techniques.

External influences: The influence of its surroundings on a built facility is modeled by specifying the values of certain variables on the surface and in the volume of the facility. For structural analysis purposes external influences are specified in terms of a body force, the static boundary conditions, i.e. prescribed values of surface tractions at given points, and kinematic boundary conditions, i.e. prescribed values of displacement variables at given points. The level of detail with which external influences are accounted for depends on the assumptions and approximations deemed to be appropriate for the given problem.

Approximation: The exact physical processes which take place when a facility reacts to environmental influences are in general too complex to be fully described and solved analytically. Approximations and assumptions regarding the motion, the material behaviour, the geometry and characteristics of the structural components[4], and the influences on the facility

[1], [2], [3] see section 1.2.3
[4] see section 2.4

are introduced to reduce the complexity. In addition, numerical algorithms are usually employed to find a solution. Consequently, structural analyses do not yield the exact values of variables which describe the behaviour of facilities. The quality of predicted behaviour depends directly on the approximations which were considered to be acceptable for the given problem, the suitability of the chosen solution algorithm, and the correct application thereof. Approximations that are often deemed to be acceptable in the analysis of built facilities have their origins in the characteristics of the building materials and the serviceability requirements imposed on such structures. The three most frequently applied idealisations are:

- Displacements are small relative to the dimensions of the components of the facility: This has the important implications that the equilibrium of the analysis model may be considered in the reference configuration, and that the strain-displacement relations are linear.
- Material is homogeneous, isotropic and behaviour is linear elastic: Under this assumption, the relationship between stress and strain is linear, and unloading reproduces the configuration before loading.
- Influences on the facility are independent of time and independent of the structural response, at least for the purposes of the analysis: This means that no changes in the boundary conditions have to be allowed for, and dynamic effects are absent.

Basic equations: For stationary media subject to small displacements, infinitesimal strain, and homogeneous, isotropic, linear elastic material behaviour, the boundary value problem is described by the following equations [Shames 1985]:

$$\left.\begin{array}{ll}
\bullet \text{ Equilibrium:} & \sigma_{ij,j} + p_{vi} = 0 \\
\bullet \text{ Strain-displacement:} & \varepsilon_{ij} = \frac{1}{2}\left(u_{i,j} + u_{j,i}\right) \\
\bullet \text{ Material (stress-strain):} & \sigma_{ij} = \lambda \varepsilon_{kk} \delta_{ij} + 2G\varepsilon_{ij} \\
\bullet \text{ Stress on surface:} & t_i = \sigma_{ij} n_j \\
\bullet \text{ Boundary conditions:} & u_i = u_{io} \text{ for } u_i \in C_u \text{ and } t_i = t_{io} \text{ for } t_i \in C_t
\end{array}\right\} \quad (2.1.1)$$

in which σ_{ij} are the components of the stress tensor, p_{vi} the components of the body force vector, ε_{ij} the linear components of the strain tensor, u_i the displacement components, λ and G the Lamé constants, t_i the components of the surface stress vector, n_j the components of the outward normal unit vector on the surface, C_u the set of prescribed displacement vector components and C_t the set of prescribed surface stress vector components.

Solution: The static, linear-elastic problem stated above is the most basic problem of structural analysis. The equation system has 18 independent equations for the 18 unknowns (6 stress, 6 strain, 3 displacement and 3 surface stress vector components). For most engineering problems, however, analytical solutions are not possible. Classical methods of structural analysis comprise a large number of ingenious techniques which were developed to solve specific problems, under the constraint that only primitive tools were available to support calculations. Their specific nature makes them unsuitable for computer implementation.

A number of numerical solution methods were developed to treat a broad class of problems. These methods are widely supported in software applications and receive attention below.

2.2 Numerical methods of structural analysis

The development of numerical techniques for the solution of mechanics problems were stimulated by the development of computers, which supplied the storage capacity and processing power required to solve the resulting equation systems. In general, numerical solutions attempt to satisfy the governing equations of the problem at certain chosen points, but not at each point of the problem domain. Three methods have found application in structural mechanics problems:

2.2.1 Finite Difference method

The Finite Difference method can be used if the differential equations are available at each point of the problem domain, and a sufficient number of boundary conditions are given on the boundary. For initial value problems, the values of behaviour variables at time t_0 must be known. The values for time $t > t_0$ are then computed.

In applications of the Finite Difference method, a grid is defined over the problem domain. At each of the grid nodes each term of the differential equation is replaced by a difference expression which references the node and its neighbours. Difference equations are obtained by substituting these expressions into the differential equations. Solution of the difference equations yield the values of the response variables at the grid nodes.

Application of the Finite Difference method is relatively easy when the problem domain can be described by a regular grid. Special modifications have to be made to define irregular boundaries and abrupt changes in material properties. In structural analysis, complicated boundary geometry is typical, which severely limits the use of the method.

2.2.2 Boundary Element method

Pre-requisites for the use of the Boundary Element method are that a sufficient number of differential equations describe the behaviour of each point in the problem domain, and that a sufficient number of boundary conditions are given. For intitial value problems, the initial values must be available, as described above.

The essential feature of the method is that an Ansatz is chosen which satisfies the governing differential equation in the domain exactly. The parameters of the trial solution are chosen in such a way that the boundary conditions are satisfied in an approximate manner. Different methods of approximation of the boundary conditions lead to different implementations of the boundary element method. The problem is formulated in terms of a boundary integral and the boundary is discretised into a network of elements, in which the behaviour is approximated with distributions of known shape, and with magnitude in terms of nodal behaviour

variables. Substitution of the approximating functions into the boundary integral, computed as the sum of element contributions, yields an equation system from which the nodal variables can be computed. The solution is carried into the domain using the chosen Ansatz.

The Boundary Element method is attractive because it reduces the dimension of the problem by one, which makes the computational mesh much simpler. The disadvantage is that the coefficient matrix of the resulting equations is fully populated, which leads to a significant computational effort compared, e.g., to the solution of profile matrices in finite element analyses. Application of the method is further limited because of the requirement that the Ansatz must satisfy the governing differential equation exactly.

2.2.3 Finite Element method

The Finite Element method[1] can be used to simulate behaviour when a sufficient number of integral equations describe the behaviour in the problem domain, and a sufficient number of boundary conditions are given. For initial value problems the initial values must be given, as described in section 2.2.1. In structural mechanics, energy principles are often used to formulate the integral equations. The Principle of virtual work for a deformable body is an example:

$$\int_C \sigma_{ij}\delta\varepsilon_{ij}\,dV = \int_C p_{vi}\delta u_i\,dV + \int_{\partial C_t} t_{i0}\delta u_i\,dA \qquad (2.2.1)$$

The above statement expresses the necessary and sufficient condition for a stress field σ_{ij} to be in equilibrium, namely that the internal virtual work (left-hand side) must be equal to the external virtual work (right-hand side) due to given body forces p_{vi} and surface tractions t_{i0}, for any kinematically compatible deformation field δu_i, $\delta\varepsilon_{ij}$. It is valid for any stress-strain law. It has been shown that additional insights can be gained by formulating the integral equations using weighted residual methods [Schutte 2000].

Application of the Finite Element method is based on subdivision of the problem domain into a mesh of finite size elements. The shape is described by the topology of the elements, the geometry of their nodes and interpolation of geometry inside, and along the edges of the elements. The physical state is interpolated in terms of generalised variables (the degrees of freedom of the finite element analysis) at chosen state points. In structural mechanics the displacement method is popular, and the degrees of freedom are usually displacement and/or rotation components. Completeness of the interpolating polynomials, and the continuity of interpolated variables within the elements and over the element boundaries, influence the accuracy and convergence properties of the method. The integral equation, describing behaviour over the problem domain, is expressed as the sum of integrals over the elements. Substitution of the interpolation functions into the element integrals yields expressions for element matrices and vectors. The integrals are computed analytically, if possible, otherwise numerically. Summation of the element contributions yields, for the finite element model, a system of algebraic equations of the form $[K]\{U\} = \{Q\}$. Given an appropriate set of bound-

[1] A complete description of the method, applied to thin plate problems, is given in section 3.2

ary conditions, the unknown variables at the state points are determined by solving the algebraic equations. Once all the variables at the state points are known, the physical state inside each element can be computed. This represents a finite element solution of the problem.

The Finite Element method's versatility and robustness are its outstanding features. Since the size and shape of elements are optional, and elements of different types and physical properties can be mixed, arbitrary geometries, loads and boundary conditions can be modeled. Furthermore the Finite Element mesh is a mathematical abstraction that is easy to visualize. As a result, the Finite Element method has become the de facto standard in the solution of structural mechanics problems. It is also the method of reference throughout this dissertation. In the following section, procedures for the analysis of built facilities are listed.

2.3 Finite Element analysis procedures

Idealisation: The natural behaviour of structural components comprises a large number of variables. In order to reduce this complexity for analysis purposes, the natural influences and behaviour are idealised in a number of ways. Idealisations leading to the linear, static problem (mentioned above) involved three basic requirements: small displacements and small strains, a linear stress-strain relationship, and boundary conditions which do not change with time or as a result of structural response. When displacements and strains become large, and/or the stress-strain relationship is non-linear, and/or boundary conditions change as a result of structural response, a non-linear analysis has to be performed. Regarding the time-independence of boundary conditions, it is clear that many of the forces acting on built facilities are functions of time, and that kinematic boundary conditions may also change with time. However, for the purposes of a specific analysis, the time-dependence of boundary conditions may often be neglected, in which case a static analysis can be performed. A dynamic analysis is required when the idealisation of time-independence of influences and behaviour is abandoned.

Main analysis procedures: Depending on the idealisations which are assumed to be applicable to the problem at hand, four main procedures of structural analysis are available, i.e. linear static , non-linear static, linear dynamic and non-linear dynamic procedures. These categories are further subdivided into specific analysis types and solution methods. Those listed below are frequently applied in the analysis of built facilities.

2.3.1 Linear analysis

A linear analysis is based on the idealisations of small displacements, linear elastic material behaviour, and non-varying boundary conditions. The material models of linear elasticity may allow, with an increasing degree of complexity, for isotropic, orthotropic or general anisotropic material conditions.

2.3.1.1 Linear static analysis

The basic assumption of static analysis procedures is that all variables describing the behaviour of the analysis model are either independent of time, that time-effects are small enough to be negligible, or that dynamic effects may be accounted for in a quasi-static way. An example of the latter is wind loading, which is often accounted for as a quasi-static load in the design of low-rise buildings, even though it clearly varies with time. The following may be computed:

- *Displacements, strains, stresses and stress resultants in elements*: In the linear case all terms in the system equations are independent of the structural response. Consequently the load versus response behaviour is linear, and a unique solution, for given boundary conditions, is reached by solving the system equations $[K]\{U\} = \{Q\}$. The element nodal

variables are extracted from the system vector, from which the element internal displacements, strains, stresses and stress resultants are interpolated.

- *Eigenvalue buckling estimates*: Actual buckling of a structure or component thereof in general involves non-linear considerations. Considering the high cost of a non-linear analysis, a linearised eigenvalue analysis may be useful since it indicates a bound to structural loading that is helpful in design.

2.3.1.2 Linear dynamic analysis

When time-effects have to be accounted for in the influences acting on a facility, a dynamic analysis has to be performed. The aim of a dynamic analysis is to determine response variables (e.g. displacements, strains, stresses and stress-resultants) in structural components as functions of time. Modification of the Principle of virtual work (eqn 2.2.1) [Shames 1985] to allow for dynamic effects and viscous energy dissipation (damping) yields:

$$\int_C \sigma_{ij}\delta\varepsilon_{ij}\,dV = -\int_C \delta u_i \rho \ddot{u}_i\,dV - \int_C \delta u_i c\dot{u}_i\,dV + \int_C p_{vi}\delta u_i\,dV + \int_{\partial C_t} t_{i0}\delta u_i\,dA \qquad (2.3.1)$$

This leads to a Finite Element equation system for the structural dynamics problem:

$$[M]\{\ddot{U}\} + [C]\{\dot{U}\} + [K]\{U\} = \{Q(t)\} \qquad (2.3.2)$$

in which $[M]$ is the system mass matrix, $[C]$ the damping matrix and $\{Q(t)\}$ the vector of loading time functions.

When the prerequisites for a linear analysis are met, the solution of these equations is usually based on modal superposition methods, in which case the following are computed:

- *Natural frequencies of the analysis model*: Consideration of the undamped free vibration conditions yields an eigenvalue problem. The extraction of a suitable number of eigenfrequencies and associated eigenvectors is the first step in a linear dynamic analysis.
- *Transient and steady-state response*: Solving the equation system (2.3.2) is problematic since the equations are coupled. For cases where the coefficient matrices are independent of the response (i.e. the linear case), the components of the response vector $\{U(t)\}$ may be expressed as a linear combination of the eigenvectors multiplied by their respective modal response time functions. Utilising the orthogonality properties of the eigenvectors, equation (2.3.2) can be transformed into an uncoupled equation system in eigenvector space if proportional damping is employed, in which the equation of motion for each mode can be solved using standard techniques. The general response time function for a mode can be computed using the Duhamel integral [Clough 1975]. A free-vibration contribution due to non-zero initial values must be added if applicable. If the loading is periodic, the steady-state response of the analysis model can be computed using a similar procedure.
- *Random response*: When external influences on a facility are not definite functions of time, but are due to random processes, the random response has to be determined. For linear systems, random response can be determined using a modal superposition technique [Clough 1975].

2.3.2 Non-linear analysis

Objective: For built facilities, the general objective is to determine the load versus displacement relation, or to find either the displacement or the load when the other is given, or to find the collapse load. When the relation between influence and response is linear, these tasks can be executed in a predetermined number of computational steps. However, when the behaviour exceeds the limitations associated with linearity, direct proportionality between influence and response in the analysis model is lost, and iterative procedures are required to achieve a solution.

Problems: Iterative procedures, usually a series of linear predictor-corrector steps, require a substantial increase in solution effort compared to linear procedures. In general, it is not possible to predict beforehand the computational effort required for a solution. A number of factors may cause the iteration to converge very slowly, or even to diverge. If convergence criteria are set too strictly the iterative process may be very expensive or may not converge at all. Criteria which are not strict enough may cause the process to diverge, or, more dangerously, to converge to the wrong answer. In modern software applications automatic incrementation schemes and convergence control are helpful, but do not relieve the engineer's responsibility to be well aquainted with the necessary mechanical principles and the fundamentals of numerical solution algorithms for non-linear mechanics. In spite of the problems, non-linear procedures are being taken up in analysis practice due to pressure on the economy of facilities, stringent design requirements and the fact that finite element algorithms and computer capacity have made it a practical possibility.

2.3.2.1 Types of non-linearity

The idealisations leading to linear procedures are linearised geometry, linear material behaviour and stable boundary conditions. When these idealisations are not valid, three types of non-linearity result, i.e. geometric non-linearity, material non-linearity and contact problems and follower forces. These are described below.

Geometric non-linearity: Geometric non-linear formulations address two problems, i.e. the problem of considering the equilibrium of structural components with respect to the deformed geometry of an analysis model, and the problem of non-linear strain-displacement relations. Since the deformed configuration is not known in advance, the solution has to be found by an iterative procedure in which the incremental form of the equilibrium equations of the system is used, i.e. $[K(U)]\{\Delta U\} = \{\Delta Q\}$, in which $[K(U)]$ depends on the displacements. This dependence can be treated systematically. [de Borst 1996] shows that the geometrically non-linear, incremental equilibrium equations for a finite element may be expressed as

$$[k_0 + k_1]_e\{\Delta u\} = \int_{C_e} [S]_e^T\{p_v\}\,dV + \int_{C_{te}} [S]_e^T\{t\}_0\,dA - \int_{C_e} [B_1]_e^T\{s\}.$$

C_e, C_{te} refer to the reference configuration of the element, $[S]$ is the element shape function matrix, and $\{s\}$ refers to instantaneous Piola-Kirchoff stress tensor components. The component $[k_0]$ contains the conventional, displacement-independent stiffness of the element.

The term $[k_1]$ contains contributions to the instantaneous stiffness matrix arising from a *linear* dependence of stiffness on the deformation. As displacements become larger, another term, $[k_2]$, accounting for quadratic dependence of stiffness on the displacements, can be added, and so on. Many Finite Element software systems implement an updating algorithm that changes the element reference coordinate system to eliminate rigid body movements. Integration is then performed referring to the updated configuration. In such cases $[k_2]$ is usually not required, and even $[k_1]$ may be neglected if strains are small.

Material: Material behaviour is observed in laboratory experiments and idealised in the form of mathematical expressions. These expressions may allow for material behaviour dependence on position, time, strain, strain-rate, etc.. In the non-linear case the material behaviour dependends on the values of variables which describe the response. If the stress-strain relation is non-linear, but elastic, there is a unique relation between stress and strain. In this case a total stress versus total strain relation may be used in a direct iteration procedure to find a point on the load-displacement curve. When plastic strains are present, however, the stress-strain relation is not unique, but depends on the loading history. An incremental theory, which relates increments of stress to increments of strain is required. The physics of material behaviour of structural elements is incorporated in structural analysis algorithms in the form of material models. For the materials used in built facilities, researchers have proposed a large variety of non-linear material models of varying complexity and purpose. Typical models important to structural analysis are listed below:

- *Plasticity models*: The behaviour of ductile metals is often described with the aid of plasticity models. The fundamental physical behaviour incorporated in plasticity models is not strictly applicable to other materials like concrete, sand, clay, and rock. For these materials, however, the incremental elastic-plastic stress strain law can be extended to allow for anisotropic plasticity, various yield criteria, alternative flow and hardening rules and the effect of cyclic inelastic loading, in which case satisfactory results may be obtained [de Borst 1996].
- *Concrete models*: Structural concrete is used extensively in built facilities, and its material behaviour is extremely complex. As a result many analyses idealise concrete behaviour as being linearly elastic or elastic-plastic. More realistic concrete models, which allow for tension cracking, post-crack response, compression crushing, and damage theory, have been developed, and implemented in commercial software packages.
- *Geotechnical models*: A number of models are available for use in geotechnical applications, e.g.the Drucker-Prager, Cam-clay and Mohr-Coulomb models. These are based on the plasticity model described above, adapted with an appropriate choice of yield condition and plastic potential function. Models for porous elasticity and permeability also exist.

- *Creep models*: Creep effects are important in the long-term evaluation of structural response, especially in concrete and wooden structures. Creep models allow for isotropic or anisotropic creep with time-hardening or strain hardening rules.
- *No compression/tension models*: These models allow materials which exhibit lack of capacity in tension or compression to be included in an analysis model. Cables and chains can, for example, not carry compressive loads, while the tensile capacity of concrete may be considered negligible in a specific analysis.
- *Hyperelastic models*: Large strain elastic response of rubberlike materials can be obtained with the aid of hyperelastic, or non-linear elastic models. There is a unique relation between stress and strain which can be determined experimentally, and implemented in a material model.

Contact problems and follower forces: Contact problems are formulated to deal with changes in kinematic boundary conditions occurring during the course of the analysis. Such highly non-linear situations arise, for example, when displacement of a point is restrained in one direction, but not in the opposite direction, or when parts of the analysis model come into contact, or lose compressive contact as a result of the deformation. Interface elements employing a special stress-strain law are sometimes used to model such behaviour. In other cases the displacements on possible contact areas are monitored and applicable displacement constraints are applied when necessary. Follower force procedures have to be applied when changes in the static boundary conditions are caused by large deformation. For example, the magnitude and/or direction of applied forces may change as the structure deforms. A follower force formulation can to some extent account for this phenomenon automatically, but in some cases the user has to specify the relationship between instantaneous geometry and forces.

2.3.2.2 Non-linear static analysis

The occurrence of any combination of the non-linearities described above, in any component or position of the analysis model, implies that a non-linear procedure has to be executed to reach a solution. For non-linear static analysis, incremental-iterative procedures, based on the Newton-Raphson method and variants thereof (i.e. tangential-stiffness methods), or quasi-Newton methods (i.e. secant-stiffness methods) [Cook 1981, de Borst 1996], are used. The basic incremental form of the system equations for static analysis is $[K(U)]\{\Delta U\} = \{\Delta Q\}$ in which $[K(U)]$ is the instantaneous stiffness matrix, and $\{\Delta U\}$, $\{\Delta Q\}$ are respectively displacement and load increments. The following may be computed:

- *Displacements, strains and stresses in elements*: In the solution procedure, increments of the loading state are applied, resulting in increments of the displacement state of the model (or vice versa). Increments are added to obtain total displacements and loads. In the elements, strains and stresses are updated in each increment. When plastic material behaviour occurs, the solution is loading-path dependent, and the prescribed loading history must be followed.

- *Collapse load:* Near the collapse load the tangential stiffnes matrix of the system becomes ill conditioned, and careful incrementation procedures are required. Arc-length methods are modified load-incrementing methods which control the incrementation along the load-displacement curve [Riks 1979]. Variants of arc-length methods have been developed which can elegantly traverse the singular points of the load-displacement curve. These methods can be used to find the nonlinear collapse load, and to study the structure's post-buckling behaviour as well.

2.3.2.3 Non-linear dynamic analysis

When some form of non-linearity is present, solutions to dynamic problems are usually found by numerical integration of the equation of motion. Finite difference expressions for the nodal velocities and accelerations are substituted into the equation, yielding a form suitable for numerical integration through time. The central issue is to maintain stability and accuracy as the integration proceeds. When the difference expressions are substituted into the equation, as written at time t, an explicit form is obtained, e.g. the central difference method [Cook 1981]. Explicit methods are attractive because they can be implemented very efficiently in computer programs [Abaqus Explicit], but care has to be taken to maintain stability. Implicit methods are obtained when the difference equations are substituted into the equation of motion written at time $t + \Delta t$, e.g. the Newmark method. Implicit methods provide better stability, but require more solution time because a linear system has to be solved in each increment. In general, available numerical integration methods can be shown to be either conditionally stable or unconditionally stable, the latter being preferred when non-linearlities are present. The displacements, strains and stresses in elements are updated as the integration proceeds.

2.4 Structural building components

As an introduction to the structural analysis problem, the equations yielded by elasticity considerations of a three-dimensional solid body are listed in equations 2.1.1. Although subject to the approximations and limitations listed in section 2.1, equations 2.1.1 is exploitable for a broad class of analyses of built facilities, since all built facilities are three-dimensional bodies. However, the cost of a full three-dimensional analysis may be high, and a more appropriate theory may yield better results, and cost less.

Dimension reduction: In many instances the geometry of a three-dimensional component of a structure may, for analysis purposes, be reasonably represented in one or two dimensions. A number of specialized building components are introduced in this way. Well known, and widely used components are:

- ***Truss and beam components:*** These are bodies whose cross-sectional dimensions are small in comparison with their length. Consequently, they are described as line elements, with their cross-sectional properties as attributes. Trusses carry only axial forces, and beams may carry a combination of axial forces, shear forces, and bending and torsional moments.
- ***Membranes and plates:*** Both membranes and plates are flat bodies, of which the thickness is small compared to their other dimensions. They are described as two dimensional elements in their plane, with thickness as an attribute. Membranes are subjected to in-plane forces, while plates are subjected to transverse loads. This gives rise to axial and in-plane shear forces in membranes, and transverse shear forces, bending and twisting moments in plates.
- ***Shells:*** Curved plates, subjected to a combination of in-plane and transverse loading, are called shells. The internal forces are also a combination of the membrane and plate actions.

Contradictions: The geometric behaviour of the reduced-dimension building components is restricted. For example, in beam bending, the Bernoulli hypothesis, which states that plane sections remain plane during deformation, is used. Similarly, in thin plate bending problems the Kirchoff hypothesis is applied. These hypotheses, based on special intuition, provide additional equations to the set 2.1.1, which lead to contradictions since there are more equations than unknowns. In the stated two examples, the transverse shear strain in the Bernoulli beam and the Kirchoff plate turn out to be zero. Such contradictions have to be accounted for.

Behaviour: For each of the specialized components, a number of theories describing their structural behaviour exist. These vary in terms of the assumptions about their initial shape, their deformed geometry, the amount of coupling between different deformations, etc. For the popular building components a large amount of research has been done, and a rich

variety of theories exist. One such theory, namely the Kirchoff theory of thin plates, is summarized in the next chapter.

Engineering semantics: Specialized building components are used extensively, and the reason for this is two-fold. Apart from reducing the number of unknowns to be solved in an analysis, the introduction of the intuitive assumptions leads to the expression of the governing equations and boundary conditions in terms of physical variables, e.g. bending moments, which engineers understand better. The relative simplicity of problem description and interpretation of results, aided by better visualization of the structure and its behaviour, contribute further to the popularity of specialized building components.

3 Finite element analysis of thin plates

The description of the finite element analysis procedures and structural building components in chapter 2 indicated the wide scope of numerical structural analysis. In production versions of distributed design software the complete scope has to be accounted for. However, for the purposes of in-depth treatment of problems associated with distributed structural analysis, the scope of the pilot implementation is limited to linear static finite element analysis of thin plates. The development of plate theory and its finite element approximation is exemplary of the treatment of other structural components. Consequently preferred solution methods indicated by this investigation can be extrapolated to the implementation of production software.

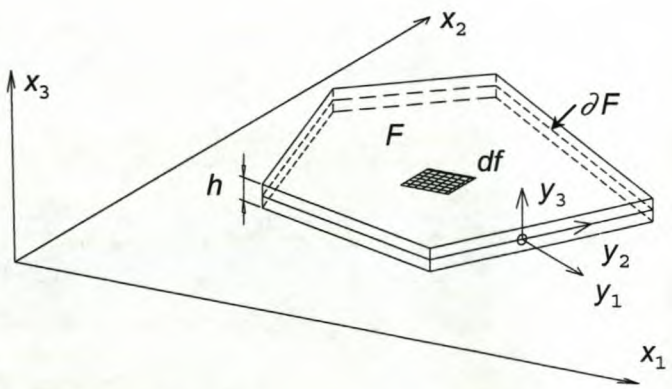
Although both topics are dealt with extensively in structural analysis literature, the Kirchoff theory of thin plates for linear static analysis, and a finite element theory for the solution of the governing equations, are summarised below [Pahl 2000]. Plate theory, and its approximation with finite elements, provide an algorithmic basis for structural analysis in local and distributed environments. These algorithms operate on objects which describe the structural system, the influences acting on the system, and the resulting behaviour of the system. The datatypes (classes) which are used to specify the components, the central parts of the finite element model, and the required user interfaces, play a significant role in the distributed analysis of such structures. They are described in chapter 4.

3.1 Theory of thin plates

The idealisation of a thin plate, its kinematics, material law, stress resultants, boundary conditions and governing equations are summarised.

3.1.1 Idealisation

Geometry: The geometry of plates is defined by considering the reference configuration C of a body. Its surface ∂C consists of a plane upper surface ∂C_u, a plane lower surface ∂C_ℓ, which is parallel to ∂C_u, and an edge surface ∂C_r which is formed by the translation of a line which is perpendicular to ∂C_u and ∂C_ℓ. The body is called a thin plate if the distance between the surfaces ∂C_u and ∂C_ℓ is small compared to the radius of the smallest circle which contains ∂C_u or ∂C_ℓ. The plane which is equidistant from ∂C_u and ∂C_ℓ is called the midsurface of the plate and is denoted by F. The intersection of the midsurface F with the edge surface ∂C_r is called the edge of the midsurface and is denoted by



∂F. The global coordinate system is chosen so that the midsurface F lies in the plane $x_3 = 0$. Consequently the upper surface ∂C_u lies in the plane $x_3 = h/2$ and the lower surface ∂C_ℓ in the plane $x_3 = -h/2$. An infinite element of the midsurface F is denoted by df, and an infinite element of the the edge ∂F by dr.

External influences: The external actions on the volume and the surface ∂C_u, ∂C_ℓ of the body are specified in the global coordinate system. The external actions on the edge surface ∂C_r are described in a local cartesian coordinate system y_1, y_2, y_3 at each point of ∂F. The axis y_1 is in the direction of the outward normal to ∂F, and the axis y_3 in the direction of the global axis x_3. For linear plate theory, it is assumed that the external actions on the body are categorised as follows:

1. Prescribed specific body force p_{v_3} parallel to axis x_3.
2. Prescribed surface stress vectors on ∂C_u and ∂C_ℓ parallel to axis x_3.
3. Prescribed stress vectors and displacements on ∂C_r, which are consistent with the assumed plate behaviour (cf. Kirchoff hypothesis below).

Kirchoff hypothesis: Due to the specific geometry and loading of the thin plate, it is assumed that the state of displacement has the following properties:

1. The displacement of all material points which lie in the midsurface of the reference configuration is normal to the midsurface.
2. Material points which lie on a straight line normal to the midsurface in the reference configuration, are displaced so that they lie on a straight line normal to the midsurface of the instant configuration.
3. The component ε_{33} of the strain tensor is zero at all points of the body.
4. The component σ_{33} of the stress tensor is zero at all points of the body.
5. The components σ_{13} and σ_{23} of the stress tensor are zero at all points of the upper and lower surfaces ∂C_u and ∂C_ℓ.

Since the Kirchoff hypothesis introduces additional equations, the number of equations is greater than the number of unknowns, which leads to contradictions in thin plate theory.

3.1.2 Kinematics

State of displacement: Let P be a point on the midsurface of the plate, and let Q be a point on the normal to the midsurface at P. Let the coordinates of P be $(x_1, x_2, 0)$, and let the coordinates of Q be (x_1, x_2, x_3). The Kirchoff hypothesis leads to the following relations between the displacements of the points P and Q:

$$\begin{Bmatrix} u_{1(Q)} \\ u_{2(Q)} \\ u_{3(Q)} \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ u_3 \end{Bmatrix} + x_3 \begin{Bmatrix} -u_{3,1} \\ -u_{3,2} \\ 0 \end{Bmatrix} \tag{3.1.1}$$

with $u_3(x_1, x_2)$ displacement of point P in the direction of axis x_3

$u_{3,i}$ partial derivative of u_3 with respect to x_i

State of strain: The state of strain at Q is determined form the state of displacement (3.1.1):

$$\begin{Bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ 2\varepsilon_{12} \end{Bmatrix} = x_3 \begin{Bmatrix} -u_{3,11} \\ -u_{3,22} \\ -2u_{3,12} \end{Bmatrix} = x_3 \{\dot{\varepsilon}\} \tag{3.1.2}$$

3.1.3 Material law

With $\sigma_{33} = 0$ due to assumption 4. of the Kirchoff hypothesis, the stress-strain relationship is

$$\begin{Bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{12} \end{Bmatrix} = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} \begin{Bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ 2\varepsilon_{12} \end{Bmatrix}$$

$$\{\sigma\} = [C]\{\varepsilon\} \tag{3.1.3}$$

3.1.4 Stress resultants

State of stress: The stress components σ_{11}, σ_{22}, σ_{12} of a thin plate are determined by substitution of the strain-displacement relations (3.1.2) into the stress-strain relations (3.1.3). The shear stress components σ_{13}, σ_{23} cannot be derived from the shear strains ε_{13}, ε_{23}, since these are zero by the Kirchoff hypothesis.

$$\{\sigma\} = [C]\{\varepsilon\} = x_3[C]\{\dot{\varepsilon}\} \tag{3.1.4}$$

$[C]$ elasticity matrix as defined in (3.1.3)

$\{\dot{\varepsilon}\}$ curvature vector defined in (3.1.2).

The state of stress is proportional to x_3. The factor of $\{\sigma\}$, which is independent of x_3, is denoted by $\{\dot{\sigma}\}$. Then, using (3.1.4) and (3.1.2):

$$\{\sigma\} = x_3\{\dot{\sigma}\} \quad \text{with} \quad \{\dot{\sigma}\} = [C]\{\dot{\varepsilon}\}$$

$$\begin{Bmatrix} \dot{\sigma}_{11} \\ \dot{\sigma}_{22} \\ \dot{\sigma}_{12} \end{Bmatrix} = \frac{E}{1-\nu^2} \begin{Bmatrix} -u_{3,11} - \nu u_{3,22} \\ -u_{3,22} - \nu u_{3,11} \\ -(1-\nu)u_{3,12} \end{Bmatrix} \tag{3.1.5}$$

Stress vectors: Cauchy's formula $t_i = \sigma_{ij} n_j$ relates the components t_i of the stress vector on an interface with unit normal components n_j to the components of the stress tensor. Considering a surface element of the face with normal $\{e\}_1^T = \{1 \quad 0 \quad 0\}$, the stress vector $\{t\}_1$, and the factor of $\{t\}_1$ which is independent of x_3, denoted by $\{\dot{t}\}_1$, are given by:

$$\{t\}_1 = \begin{Bmatrix} \sigma_{11} \\ \sigma_{12} \\ 0 \end{Bmatrix} = x_3 \begin{Bmatrix} \dot{\sigma}_{11} \\ \dot{\sigma}_{12} \\ 0 \end{Bmatrix} = x_3 \{\dot{t}\}_1 \tag{3.1.6}$$

Analogously, the vector acting on the face with normal $\{e\}_2^T = \{0 \quad 1 \quad 0\}$, denoted by $\{t\}_2$, and its factor which is independent of x_3, denoted by $\{\dot{t}\}_2$ is:

$$\{t\}_2 = \begin{Bmatrix} \sigma_{21} \\ \sigma_{22} \\ 0 \end{Bmatrix} = x_3 \begin{Bmatrix} \dot{\sigma}_{21} \\ \dot{\sigma}_{22} \\ 0 \end{Bmatrix} = x_3 \{\dot{t}\}_2 \tag{3.1.7}$$

Force stress resultants: A volume element of the plate, with edges parallel to the global coordinate axes, is considered. The element extends over the full height h of the plate, and its sidelenghts in the midsurface are $dx_1 = dx_2 = 1$. On the face with normal $\{e\}_1$, the force vector acting on the element with area $dx_1 dx_3 = 1 dx_3$, is $\{t\}_1 dx_3$. The force stress resultant on the face is found by integration over the plate thickness, using (3.1.6):

$$\int_{-h/2}^{+h/2} \{t\}_1 dx3 = \int_{-h/2}^{+h/2} x_3 \{\dot{t}\}_1 dx3 = \{0\} \tag{3.1.8}$$

The force stress resultant on the face with normal $\{e\}_2$ is found analogously:

$$\int_{-h/2}^{+h/2} \{t\}_2 dx3 = \int_{-h/2}^{+h/2} x_3 \{\dot{t}\}_2 dx3 = \{0\} \tag{3.1.9}$$

The bending of thin plates does not lead to force stress resultants in the midsurface. (This result is restricted to linear plate theory).

Moment stress resultants: The moment of the force $\{t\}_1 dx_3$ about a point P in the midsurface is $\{\{e\}_3 x_3\} \times \{\{t\}_1 dx_3\}$. The moment stress resultant $\{m\}_1$ on the face with normal $\{e\}_1$ is found by integration using (3.1.6):

$$\{m\}_1 = \int_{-h/2}^{+h/2} \begin{Bmatrix} 0 \\ 0 \\ 1 \end{Bmatrix} \times \begin{Bmatrix} \sigma_{11} \\ \sigma_{12} \\ 0 \end{Bmatrix} x_3 \, dx_3 = \int_{-h/2}^{+h/2} \begin{Bmatrix} -\dot{\sigma}_{12} \\ \dot{\sigma}_{11} \\ 0 \end{Bmatrix} x_3^2 \, dx_3 = \frac{h^3}{12} \begin{Bmatrix} -\dot{\sigma}_{12} \\ \dot{\sigma}_{11} \\ 0 \end{Bmatrix} \tag{3.1.10}$$

$\{m\}_1$ moment stress resultant on the face with normal $\{e\}_1$.

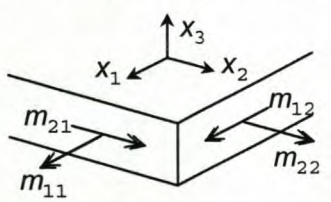
The moment stress resultant on the face with normal $\{e\}_2$ is found analogously:

$$\{m\}_2 = \int_{-h/2}^{+h/2} \begin{Bmatrix} 0 \\ 0 \\ 1 \end{Bmatrix} \times \begin{Bmatrix} \sigma_{21} \\ \sigma_{22} \\ 0 \end{Bmatrix} x_3 \, dx_3 = \int_{-h/2}^{+h/2} \begin{Bmatrix} -\dot{\sigma}_{22} \\ \dot{\sigma}_{21} \\ 0 \end{Bmatrix} x_3^2 \, dx_3 = \frac{h^3}{12} \begin{Bmatrix} -\dot{\sigma}_{22} \\ \dot{\sigma}_{21} \\ 0 \end{Bmatrix} \tag{3.1.11}$$

$\{m\}_2$ moment stress resultant on the face with normal $\{e\}_2$.

The component of the moment $\{m\}_k$ for the basis vector $\{e\}_i$ is denoted by m_{ik}. The moment vectors on the faces of the volume element are thus:

$$\{m\}_1 = \begin{Bmatrix} m_{11} \\ m_{21} \\ 0 \end{Bmatrix} \qquad \{m\}_2 = \begin{Bmatrix} m_{12} \\ m_{22} \\ 0 \end{Bmatrix} \tag{3.1.12}$$



Shear stresses: Due to the Kirchoff hypothesis, the shear stresses σ_{31} and σ_{32} cannot be derived from the shear strains ε_{31} and ε_{32}. They are instead determined using the equilibrium equations,for zero body forces:

$$\sigma_{13,3} = -\sigma_{11,1} - \sigma_{21,2} \quad \text{and} \quad \sigma_{23,3} = -\sigma_{12,1} - \sigma_{22,2} \tag{3.1.13}$$

According to assumption 5. of the Kirchoff hypothesis (cf. p3-2), the shear stress at the lower surface

∂C_ℓ is zero. Integrating (3.1.13) between the lower surface and $x_3 = t$, over faces of unit width and normals respectively $\{e\}_1$ and $\{e\}_2$ yield:

$$\sigma_{13} = -\frac{6}{h^3}\left(t^2 - \frac{h^2}{4}\right)\left(\frac{\partial m_{21}}{\partial x_1} + \frac{\partial m_{22}}{\partial x_2}\right) \quad \text{and} \quad \sigma_{23} = +\frac{6}{h^3}\left(t^2 - \frac{h^2}{4}\right)\left(\frac{\partial m_{11}}{\partial x_1} + \frac{\partial m_{12}}{\partial x_2}\right) \tag{3.1.14}$$

Shear stress resultants: The integral of the shear stress σ_{13} over the face with normal $\{e\}_1$ is called the shear stress resultant and is denoted by q_{13}. Then, also using (3.1.5):

$$q_{13} = \int_{-h/2}^{+h/2} \sigma_{13}\, dt = \frac{\partial m_{21}}{\partial x_1} + \frac{\partial m_{22}}{\partial x_2} = -\frac{Eh^3}{12(1-\nu^2)}\frac{\partial}{\partial x_1}\left(u_{3,11} + u_{3,22}\right) \tag{3.1.15}$$

Analogously, the shear stress resultant q_{23} on the face with normal $\{e\}_2$ is:

$$q_{23} = \int_{-h/2}^{+h/2} \sigma_{23}\, dt = -\frac{\partial m_{11}}{\partial x_1} - \frac{\partial m_{12}}{\partial x_2} = -\frac{Eh^3}{12(1-\nu^2)}\frac{\partial}{\partial x_2}\left(u_{3,11} + u_{3,22}\right) \tag{3.1.16}$$

3.1.5 Boundary conditions

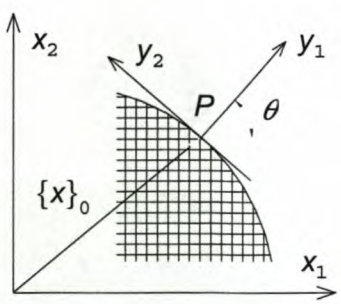
Load on the midsurface: The loads on the volume C, the upper surface ∂C_u and the lower surface ∂C_ℓ of the plate act parallel to the axis x_3:

$x \in C$	:	$p_{v1} = p_{v2} = 0$	p_{v3}	specified
$x \in \partial C_u$	:	$t_{10} = t_{20} = 0$	$t_{30} := t_{3u}$	specified
$x \in \partial C_\ell$	:	$t_{10} = t_{20} = 0$	$t_{30} := t_{3\ell}$	specified

In the idealised plate model, these loads are replaced by a load per unit area of the midsurface F, which acts parallel to the axis x_3 and is denoted by p_f:

$$\{x\} \in F: \qquad p_{f1} = p_{f2} = 0 \qquad p_{f3} := p_f \quad \text{specified}$$
$$p_f = t_{3u} + t_{3\ell} + \int_{-h/2}^{+h/2} p_{v3}\, dx_3 \tag{3.1.17}$$

Local coordinate system: The boundary conditions of the edge surface ∂C_r of the plate are specified in a local cartesian coordinate system y_1, y_2, y_3. The axis y_1 is in the direction of the outward normal to the edge ∂F. The axis y_3 is parallel to the axis x_3.



$$\{x\} = \{x\}_0 + [R]\{y\}$$
$$\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} x_{10} \\ x_{20} \\ x_{30} \end{Bmatrix} + \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} y_1 \\ y_2 \\ y_3 \end{Bmatrix} \tag{3.1.18}$$

Kinematic boundary conditions: Due to the Kirchoff hypothesis, only the local displacement component v_3 and its partial derivative with respect to y_1, $v_{3,1}$, can be specified on the edge ∂F. It is convenient to replace the partial derivatives of the displacement v_3 at a point P by the rotation vector $\{\beta\}$, with the local components $\beta_1 = v_{3,2}$, $\beta_2 = -v_{3,1}$, $\beta_3 = 0$. The

rotation β_i about axis y_i is positive by the corkscrew rule. The kinematic boundary conditions are then specified as follows:

$$\begin{aligned} v_3 &= v_{30} \\ \beta_2 &= \beta_{20} \end{aligned} \tag{3.1.19}$$

Static boundary conditions: The local components s_i of the stress vector $\{s\}$ at a point P on the edge surface ∂C_r vary with y_3, in analogy to (3.1.6):

$$\{s\}_Q = \{s\} + y_3\{\dot{s}\} \tag{3.1.20}$$

$$\begin{Bmatrix} s_{1(Q)} \\ s_{2(Q)} \\ s_{3(Q)} \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ s_3 \end{Bmatrix} + y_3 \begin{Bmatrix} \dot{s}_1 \\ \dot{s}_2 \\ 0 \end{Bmatrix}$$

s_3 average shear stress on the face normal to axis y_1 (cf. (3.1.14))

$\dot{s}_i$ factor of s_i which is independent of y_3

The integration of the local stresses over the height of the plate leads to a shear stress resultant $\{q\}_3$:

$$\{q\}_3 = \int_{-h/2}^{+h/2} (\{s\} + y_3\{\dot{s}\})\, dy_3 = h\{s\} = h \begin{Bmatrix} 0 \\ 0 \\ s_3 \end{Bmatrix} \tag{3.1.21}$$

Integration over the height of the plate, of the moment of the local stresses about the midsurface, leads to moment stress resultant $\{m\}$ with components m_1 and m_2 about the axes y_1 and y_2:

$$\{m\} = \int_{-h/2}^{+h/2} \begin{Bmatrix} 0 \\ 0 \\ 1 \end{Bmatrix} \times \begin{Bmatrix} \dot{s}_1 \\ \dot{s}_2 \\ 0 \end{Bmatrix} y_3^2\, dy_3 = \begin{Bmatrix} m_1 \\ m_2 \\ 0 \end{Bmatrix} = \frac{h^3}{12} \begin{Bmatrix} -\dot{s}_2 \\ \dot{s}_1 \\ 0 \end{Bmatrix} \tag{3.1.22}$$

The static boundary conditions at a point P on the edge ∂F of the midsurface of the plate are thus:

$$\begin{aligned} m_{y1} &= m_{y10} \\ m_{y2} &= m_{y20} \\ q_3 &= q_{30} \end{aligned} \tag{3.1.23}$$

The Kirchoff hypothesis leads to two kinematic boundary conditions (3.1.19), but to three static boundary conditions (3.1.23). This contradiction is resolved by replacing the torsional moment m_{y10} and the shear load q_{30} by an equivalent shear load $\overline{q}_{30}$:

$$\overline{q}_{30} = q_{30} + \frac{\partial m_{10}}{\partial y_2} \tag{3.1.24}$$

The kinematic boundary conditions (3.1.19) are then associated with the static boundary conditions:

$$\begin{aligned} m_{y2} &= m_{y20} \\ q_3 &= \overline{q}_{30} \end{aligned} \tag{3.1.25}$$

3.1.6 Integral governing equations

A general integral equation for linear elastic theory of three dimensional bodies is given by [Pahl 2000]:

$$\int_C \delta\{\varepsilon\}^T \{\sigma\}\, dV = \int_C \delta\{u\}^T \{p_v\}\, dV + \int_{C_t} \delta\{u\}^T \{t\}_0\, dA + \int_{C_u} \delta\{u\}^T \{t\}\, dA \qquad (3.1.26)$$

$u_i \in C_u \quad \Rightarrow \quad u_i = u_{i0}$

C_u set of prescribed displacement components on ∂C

C_t set of prescribed stress components on ∂C

The above equation is an extension of the Principle of virtual work equation (2.2.1), by the addition of the integral of the unknown stress vector components over C_u. Integration over the thickness of the plate yields the integral governing equation of thin plate theory:

$$\int_F \frac{h^3}{12} \delta\{\dot{\varepsilon}\}[C]\{\dot{\varepsilon}\}\, df = \int_F \delta u_3 p_f\, df + \int_{C_t} \delta\{v\}^T \{q\}_0\, dr + \int_{C_u} \delta\{v\}^T \{q\}\, dr \qquad (3.1.27)$$

in which $\{v\} = \begin{Bmatrix} v_3 \\ \beta_1 \\ \beta_2 \end{Bmatrix}, \quad \{q\} = \begin{Bmatrix} q_3 \\ m_1 \\ m_2 \end{Bmatrix}$

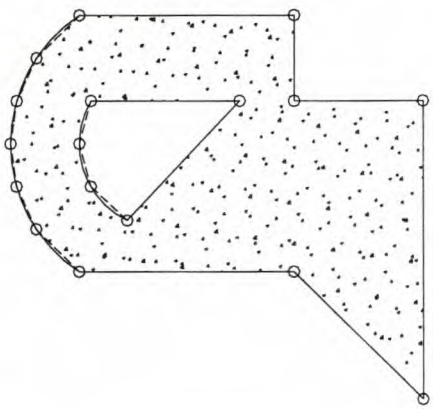
The kinematic boundary conditions (3.1.19) must be satisfied for displacement components $v_3, \beta_1, \beta_2 \in C_u$:

$$\{v\} \in C_u \quad \Rightarrow \quad \{v\} = \{v\}_0 \qquad (3.1.28)$$

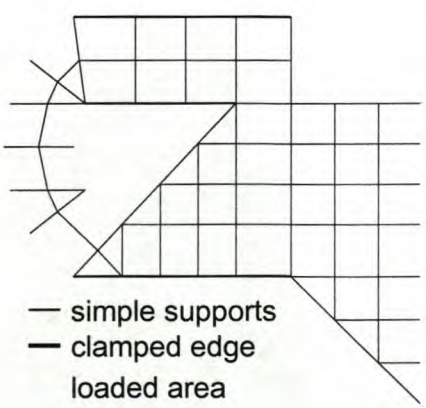
3.2 Finite Element theory of thin plates

Analytical solutions to the governing equations of structural components are only available in cases where the components have highly regular shapes, e.g. rectangular or circular, and the boundary conditions are also regular, e.g. loading on a full surface or at the centre of a circular surface. However, in structural analysis practice the shape and boundary conditions of components are generally irregular.The finite element method addresses this problem through piecewise approximation of the geometry and physical state of components using finite size elements. Finite element theory of thin plates is described below. For components other than plates, the details regarding element shapes, degrees of freedom, interpolation, governing equations, etc. are different, but the procedure of developing a finite element theory for them is similar.

Geometric approximation: The midsurface of a plate frequently has an irregular shape, e.g. floor slabs in buildings and bridge decks. In general, the outer edge of a plate may be composed of straight and curved segments, meeting at different angles, and inside the plate there may be openings which are irregularly shaped as well. It is convenient to approximate the edges of the midsurface by straight line polygons, which are readily described by their topology and the coordinates of their nodes.



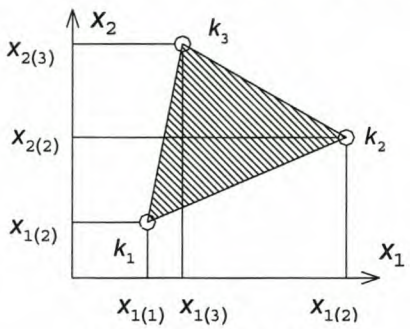
Physical approximation: The midsurface of the plate is supported at specified points and along specified lines. It may be partially loaded. In order to describe such external actions, it is convenient to decompose the midsurface of the plate into elements of finite size, for instance into triangles and quadrilaterals. The nodes and edges of these finite elements are placed to coincide with the points and lines where displacements are prescribed, and with the edges of loaded areas.



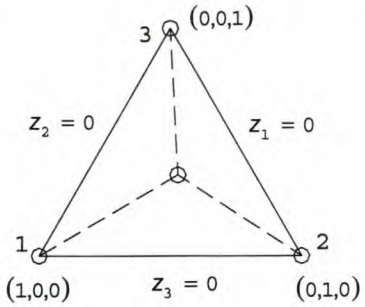
In order to solve the integral governing equations for the behavioural variables such as the displacements of the midsurface of the plate, Ansätse for the variation of these variables over the midsurface must be chosen. For plates with irregular shape, suitable functions of the location vector $\{x\}$ are not available as Ansätse. Therefore the entire midsurface is decomposed into finite elements. For each of these elements, simple Ansatz functions are available, whose value outside the element is zero. Consequently, the total Ansatz for the plate is the sum of the Ansätse over all its elements.

3.2.1 Finite geometry

Element shape: A network of triangles can be used to approximate arbitrary planar geometry. For plates, consequently, triangular finite elements are popular. In cases where the plates have strictly rectangular shape and boundary conditions, it can be advantageous to use rectangular finite plate elements. Combinations of triangular and rectangular elements are often used. Finite element theory of thin plates, for triangular elements, is summarised below.



Global coordinates



Normalised coordinates

In the global coordinate system the coordinates $x_{1(i)}$, $x_{2(i)}$ of the nodes k_i of a specific triangle are given. In the normalised coordinate system, the triangular coordinates $(1,0,0)$, $(0,1,0)$, $(0,0,1)$ of the nodes i are the same for all triangles. The normalised coordinates of an arbitrary point P of the triangle are denoted by (z_1, z_2, z_3). There are 2 independent global coordinates x_1, x_2, but 3 normalised coordinates z_1, z_2, z_3. However, the normalised coordinates are not independent, but are related by:

$$z_1 + z_2 + z_3 = 1 \tag{3.2.1}$$

Topological mapping: In the triangulation of the midsurface of the plate, the nodes are identified by natural numbers. The topology of a specific triangle is specified by its node numbers k_1, k_2, k_3. The normal of the triangular surface is placed in the direction of the axis x_3 by a right-hand corkscrew rule on the nodal sequence. In the normalised coordinate system, the nodes of the triangle are always labelled as $1, 2, 3$ in the same sequence as the element topology. Consequently, the relationship between the global and the normalised labels is described with a topological mapping: $t : K \rightarrow N$ with $t(k_1, k_2, k_3) = (1, 2, 3)$

Geometrical interpolation: The shape of a finite element is determined by the coordinates of its nodes and the interpolation functions for the coordinates. The interpolation functions determine the geometry of the edges of the element. For the triangular element with 3 nodes, the interpolation functions are linear, consequently the edges are straight. The interpolation functions are written in terms of the normalised coordinates:

$$\begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \begin{bmatrix} x_{1(1)} & x_{1(2)} & x_{1(3)} \\ x_{2(1)} & x_{2(2)} & x_{2(3)} \end{bmatrix} \begin{Bmatrix} z_1 \\ z_2 \\ z_3 \end{Bmatrix} = [X]\{s\} \tag{3.2.2}$$

$[X]$ node coordinate matrix

$\{s\}$ shape vector of linear shape functions z_1, z_2, z_3

Expressions for derivatives: For the special case of the linear triangle, the following relations can be developed:

- Normalised derivatives of the shape vector:

$$[S]_z = \begin{bmatrix} s_{1,z_1} & s_{1,z_2} & s_{1,z_3} \\ s_{2,z_1} & s_{2,z_2} & s_{2,z_3} \\ s_{3,z_1} & s_{3,z_2} & s_{3,z_3} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.2.3}$$

- Normalised derivatives of the global coordinates:

$$\begin{Bmatrix} dx_1 \\ dx_2 \end{Bmatrix} = \begin{bmatrix} x_{1(1)} & x_{1(2)} & x_{1(3)} \\ x_{2(1)} & x_{2(2)} & x_{2(3)} \end{bmatrix} \begin{Bmatrix} dz_1 \\ dz_2 \\ dz_3 \end{Bmatrix} \tag{3.2.4}$$

- Independent normalised derivatives:

$$\begin{Bmatrix} dx_1 \\ dx_2 \end{Bmatrix} = \begin{bmatrix} (x_{1(1)} - x_{1(3)}) & (x_{1(2)} - x_{1(3)}) \\ (x_{2(1)} - x_{2(3)}) & (x_{2(2)} - x_{2(3)}) \end{bmatrix} \begin{Bmatrix} dz_1 \\ dz_2 \end{Bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} \begin{Bmatrix} dz_1 \\ dz_2 \end{Bmatrix} \tag{3.2.5}$$

- Global derivatives of the normalised coordinates:

$$[Z]_x = \begin{bmatrix} z_{1,1} & z_{1,2} \\ z_{2,1} & z_{2,2} \\ z_{3,1} & z_{3,2} \end{bmatrix} = \begin{bmatrix} \{d\}_1 & \{d\}_2 \end{bmatrix} = \frac{1}{c_{11}c_{22} - c_{12}c_{21}} \begin{bmatrix} c_{22} & -c_{12} \\ -c_{21} & c_{11} \\ c_{21} - c_{22} & c_{12} - c_{11} \end{bmatrix} \tag{3.2.6}$$

- Global derivatives of the shape vector:

$$[S]_x = \begin{bmatrix} s_{1,1} & s_{1,2} \\ s_{2,1} & s_{2,2} \\ s_{3,1} & s_{3,2} \end{bmatrix} = \begin{bmatrix} \{g\}_1 & \{g\}_2 \end{bmatrix} = \frac{1}{c_{11}c_{22} - c_{12}c_{21}} \begin{bmatrix} c_{22} & -c_{12} \\ -c_{21} & c_{11} \\ c_{21} - c_{22} & c_{12} - c_{11} \end{bmatrix} \tag{3.2.7}$$

3.2.2 Finite physical state

3.2.2.1 Description of the physical state

State variables: The primary state variable for plate bending is the displacement u_3 of the midsurface. Once u_3 is known as a function of the location $\{x\}$ on the midsurface, all other state variables can be determined as functions of u_3, using the expressions derived in section 3.1.

Nodal variables: The values of the state values and their derivatives at specified nodes of the finite element net are chosen as variables of the finite element analysis. These variables are either specified as boundary conditions, or determined by solution of the governing equations. Between the nodes, the values of the state variables are determined by interpolation in the finite elements.

Variables at internal nodes: At each node of the finite element network, the displacement u_3 and the global coordinates α_1, α_2 of the rotation vector are introduced as nodal variables. The variables at node i are collected in a nodal displacement vector $\{u\}_{n(i)}$:

$$\{u\}_{n(i)} = \begin{Bmatrix} u_3 \\ \alpha_1 \\ \alpha_2 \end{Bmatrix} \tag{3.2.8}$$

u_3 displacement of the midsurface in direction x_3

α_j rotation about axis x_j (corkscrew rule).

The rotations α_i can be expressed as derivatives of the displacement u_3:

$$\alpha_1 = \frac{\partial u_3}{\partial x_2} \qquad \alpha_2 = -\frac{\partial u_3}{\partial x_1} \tag{3.2.9}$$

The nodal displacement vectors of the three nodes of an element are arranged in an element displacement vector $\{u\}_e$:

$$\begin{aligned} \{u\}_e^T &= \{u_{3(1)} \quad \alpha_{1(1)} \quad \alpha_{2(1)} \mid u_{3(2)} \quad \alpha_{1(2)} \quad \alpha_{2(2)} \mid u_{3(3)} \quad \alpha_{1(3)} \quad \alpha_{2(3)}\} \\ &= \{\{u\}_{n(1)}^T \quad \{u\}_{n(2)}^T \quad \{u\}_{n(3)}^T\} \end{aligned} \tag{3.2.10}$$

Variables at edge nodes: On the edge of the plate, either displacements or stress resultants can be specified as boundary conditions. Therefore nodal vectors are defined for both displacements and stress resultants. The edge node displacement vector $\{v\}_{n(i)}$ contains the displacement v_3 and the local components β_1, β_2 of the rotation vector at node i. The edge node stress resultant vector $\{q\}_{n(i)}$ contains the shear stress resultant q_3 and the local components m_1, m_2 of the moment stress resultant at node i:

$$\{v\}_{n(i)} = \begin{Bmatrix} v_3 \\ \beta_1 \\ \beta_2 \end{Bmatrix} \qquad \{q\}_{n(i)} = \begin{Bmatrix} q_3 \\ m_1 \\ m_2 \end{Bmatrix} \tag{3.2.11}$$

v_3 displacement of the midsurface in direction $y_3 = x_3$

β_i rotation about axis y_i

q_3 shear stress resultant in the direction y_3

m_i moment about axis y_i

Transformation of the nodal displacement vectors: The nodal displacement vector of a node on the edge of the plate is denoted by $\{u\}_n$ if the global displacement components are specified, and by $\{v\}_n$ if its local components are specified. The displacement components u_3 and v_3 are equal. The relationship between the global and local components of the rotation vector is determined with the coordinate transformation:

$$\begin{Bmatrix} u_3 \\ \alpha_1 \\ \alpha_2 \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \begin{Bmatrix} v_3 \\ \beta_1 \\ \beta_2 \end{Bmatrix}$$

$$\{u\}_n = [R]_n \{v\}_n \tag{3.2.12}$$

3.2.2.2 Internal interpolation

Interpolation of transverse displacement: The triangular element has 9 degrees of freedom. It is not possible to find an interpolation function for u_3 by inspection. The interpolation functions for the displacement u_3 are chosen in the normalised coordinate system. The free parameters are related to the nodal displacements and their normalised derivatives. These nodal variables are then expressed in terms of the nodal displacement vectors $\{u\}_{n(i)}$ and the element displacement vector $\{u\}_e$. The following result is obtained:

$$\begin{aligned} u_3 &= \{f\}^T [V]_e [A]_e \{u\}_e \\ &= \{f\}^T [P]_e \{u\}_e \\ &= [S]_e \{u\}_e \end{aligned} \tag{3.2.13}$$

in which:

$[S]_e$ displacement interpolation matrix

$\{f\}$ form vector, with

$$\{f\}^T = \left\{ z_1 \;\vdots\; z_2 \;\vdots\; z_3 \;\vdots\; z_2 z_3 \;\vdots\; z_3 z_1 \;\vdots\; z_1 z_2 \;\vdots\; (z_2 - z_3)^3 \;\vdots\; (z_3 - z_1)^3 \;\vdots\; (z_1 - z_2)^3 \right\} \tag{3.2.14}$$

$\{u\}_e$ nodal displacement vector, see (3.2.10)

$[P]_e = [V]_e [A]_e$, the form matrix of element e. (3.2.15)

$[V]_e$ a matrix of constants (determined explicitly)

$[A]_e$ a matrix containing the coefficients $c_{\alpha\gamma}$ of equation (3.2.5) ($\alpha, \gamma = 1,2$)

3.2.2.3 Interpolation of strain

Global 1st derivatives: The global derivatives of an arbitrary element f_i of the interpolation vector $\{f\}$ are determined with the chain rule:

$$\frac{\partial f_i}{\partial x_k} = \frac{\partial f_i}{\partial z_1}\frac{\partial z_1}{\partial x_k} + \frac{\partial f_i}{\partial z_2}\frac{\partial z_2}{\partial x_k} + \frac{\partial f_i}{\partial z_3}\frac{\partial z_3}{\partial x_k} \qquad k = 1, 2 \tag{3.2.16}$$

The normalised derivatives of the form vector, $\partial\{f\}/\partial z_j$, are readily determined from equation (3.2.14). The global derivatives of the normalised coordinates, $\partial\{z\}/\partial x_k$, are given by equation (3.2.6). For the special case of the linear triangle, the form matrix $[P]_e$ is constant inside the element. The global 1st derivative of u_3 follows from (3.2.13):

$$\frac{\partial u_3}{\partial x_k} = \frac{\partial \{f\}^T}{\partial x_k}[P]_e\{u\}_e \tag{3.2.17}$$

Global 2nd derivatives: The global 2nd derivatives of an arbitrary element f_i of the interpolation vector $\{f\}$ are determined with the chain rule:

$$\frac{\partial^2 f_i}{\partial x_k \partial x_m} = \sum_r \sum_s \frac{\partial^2 f_i}{\partial z_r \partial z_s}\frac{\partial z_r}{\partial x_k}\frac{\partial z_s}{\partial x_m} \tag{3.2.18}$$

The normalised 2nd derivatives of the form vector, $\partial^2\{f\}/\partial z_r \partial z_s$, are readily determined from equation (3.2.14). The global derivatives of the normalised coordinates, $\partial\{z\}/\partial x_k$, are given by equation (3.2.6). For the special case of the linear triangle, the form matrix $[P]_e$ is constant inside the element. The global 2nd derivative of u_3 follows from (3.2.13):

$$\frac{\partial^2 u_3}{\partial x_k \partial x_m} = \frac{\partial^2 \{f\}^T}{\partial x_k \partial x_m}[P]_e\{u\}_e \tag{3.2.19}$$

Strain interpolation: The strain interpolation matrix $[B]_e$ is determined by substitution of (3.2.19) into the strain-displacement relations (3.1.2):

$$\{\varepsilon\} = x_3\begin{Bmatrix} -\{f\}^T_{,11} \\ -\{f\}^T_{,22} \\ -2\{f\}^T_{,12} \end{Bmatrix}[P]_e\{u\}_e = x_3[B]_e\{u\}_e \tag{3.2.20}$$

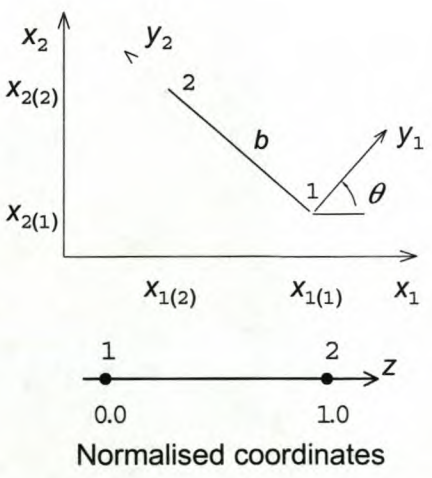
3.2.2.4 Edge interpolation

Geometry: It is assumed that the edges of the midsurface of the plate are polygons consisting of straight lines. Each line between two edge nodes is treated as a finite element. The element is described in a local cartesian coordinate system y, as defined in (3.2.1).

The element is also described in a normalised coordinate system z. The normalised coordinate z is mapped to the local coordinate y_2:

$$y_2 = bz, \quad dy_2 = b\,dz \tag{3.2.21}$$

b length of the edge in the local system



Normalised coordinates

Displacement interpolation: The displacement vector on the line element is denoted by $\{w\}$. Its coordinates are the displacement w_3 in the direction of axis y_3, and the coordinates γ_1, γ_2 of the rotation vector referred to the local coordinate system of the line element. The value of $\{w\}$ at node i is denoted by $\{w\}_{n(i)}$. The vectors $\{w\}_{n(1)}$ and $\{w\}_{n(2)}$

are arranged in an element nodal displacement vector $\{w\}_r$. The transverse displacement along the edge, w_3, is interpolated with Hermite polynomials. The rotation γ_1 is the derivative of w_3 with respect to y_2. The rotation γ_2 is interpolated linearly. The state of displacement of a point z on the line element is thus:

$$\{w\} = [S]_r \{w\}_r \tag{3.2.22}$$

$$\{w\}^T = \{w_3 \quad \gamma_1 \quad \gamma_2\}$$

$$[S]_r = \left[\begin{array}{c:c:c|c:c:c} 1-3z^2+2z^3 & bz(1-z)^2 & 0 & z^2(3-2z) & -bz^2(1-z) & 0 \\ \hdashline \tfrac{6}{b}z(z-1) & 1-4z+3z^2 & 0 & \tfrac{6}{b}z(z-1) & -2z+3z^2 & 0 \\ \hdashline 0 & 0 & 1-z & 0 & 0 & z \end{array}\right]$$

$$\{w\}_r^T = \{w_{3(1)} \quad \gamma_{1(1)} \quad \gamma_{2(1)} \mid w_{3(2)} \quad \gamma_{1(2)} \quad \gamma_{2(2)}\}$$

Stress resultant interpolation: The stress resultant vector on the line element is denoted by $\{q\}$. Its coordinates are the shear stress resultant q_3 in the direction of axis y_3, and the components m_1, m_2, referred to the local coordinate system of the line element, of the moment stress resultant. The value of $\{q\}$ at node i is denoted by $\{q\}_{ni}$. The vectors $\{q\}_{n1}$ and $\{q\}_{n2}$ are arranged in an element nodal stress resultant vector $\{q\}_r$. All stress resultants are interpolated linearly on the edge. The stress resultant state of a point z on the line element is thus:

$$\{q\} = [S]_q \{q\}_r \tag{3.2.23}$$

$$\{q\}^T = \{q_3 \quad m_1 \quad m_2\}$$

$$[S]_q = \left[\begin{array}{c:c:c|c:c:c} 1-z & 0 & 0 & z & 0 & 0 \\ \hdashline 0 & 1-z & 0 & 0 & z & 0 \\ \hdashline 0 & 0 & 1-z & 0 & 0 & z \end{array}\right]$$

$$\{q\}_r^T = \{q_{3(1)} \quad m_{1(1)} \quad m_{2(1)} \mid q_{3(1)} \quad m_{1(2)} \quad m_{2(2)}\}$$

3.2.3 Governing equations

3.2.3.1 Finite Element network

Concept: The finite element method is an approximate method for the solution of the governing equations (3.1.27) for thin plates. The midsurface of the plate is covered by a network of triangular elements. The edges of the plate are covered by line elements. The governing equation (3.1.27) contains integrals over the midsurface F and over the edge ∂F. The contribution of a particular element of the finite element net to these integrals is restricted to the domain of the element itself, since the element interpolation functions are zero outside the element. It is therefore convenient to replace the integral over F by the sum of the inte-

grals over the triangular element areas F_e, and to replace the integral over ∂F by the sum of the integrals over the line elements L_r:

$$\sum_e \int_{F_e} \frac{h^3}{12} \delta\{\dot{\varepsilon}\}^T [C] \{\dot{\varepsilon}\} df = \sum_e \int_{F_e} \delta u_3 p_f \, df + \sum_{C_t} \int_{L_r} \delta\{w\}^T \{q\}_0 dr + \sum_{C_u} \int_{L_r} \delta\{w\}^T \{q\} dr \qquad (3.2.24)$$

The interpolation functions for the displacements, the strains and the stress resultants, are substituted into the element integrals. Numerical integration over the triangular elements leads to element stiffness matrices and load vectors, which are assembled into a system stiffness matrix and a system load vector. Integration over the line elements leads to concentrated shear forces and moments at the nodes, which are added to the system load vector if they are prescribed, or are entered in the reaction vector of the system equations if they are unknown. The algebraic system equations are solved for the unknown nodal variables:

$$[K]_s \{u\}_s = \{q\}_s + \{r\}_s \qquad (3.2.25)$$

$[K]_s$ system stiffness matrix

$\{u\}_s$ system displacement vector

$\{q\}_s$ system load vector

$\{r\}_s$ system reaction vector

System vectors: Neighbouring elements posses common nodal displacement vectors. In order to describe this relationship, the nodal displacement variables of the network are numbered sequentially and arranged in a system displacement vector $\{u\}_s$ with elements u_{si}. Area loads on the triangular elements, line loads on the line elements, and concentrated node loads lead to a load vector $\{q\}_s$ with equal dimension and elements q_{si}. Neighbouring line elements posses common supports. Their reactions are arranged in a reaction vector $\{r\}_s$ with components r_{si}. If a displacement coordinate at a node is prescribed, the corresponding row of $\{r\}_s$ contains an unknown reaction, which is determined by solution of the system equations. All other coefficients of $\{r\}_s$ are zero.

Local coordinate systems: At any node of the network, a local coordinate system may be chosen. The coordinates of the nodal displacements and forces in the vectors (3.2.25) are then referred to the local system. If a local coordinate system is not specified, the coordinates in (3.2.25) are referred to the global coordinate system. The coordinates are transformed with equation (3.2.12).

3.2.3.2 Contribution of the triangular elements

Element stiffness matrix: The term on the left hand side of equation (3.2.24) is considered. Interpolation functions for the system state of strain $\{\dot{\varepsilon}\}$ and the Galerkin variation with respect to u_{si}, $\delta\{\dot{\varepsilon}\}$, are developed in terms of the system displacement vector $\{u\}_s$:

$$\{\dot{\varepsilon}\} = \sum_e [B]_e [R]_e [T]_e \{u\}_s \tag{3.2.26}$$

$$\delta\{\dot{\varepsilon}\}_i = \sum_e [B]_e [R]_e [T]_e \{e\}_i \qquad i \in \{1,\ldots,n\} \tag{3.2.27}$$

in which

$[B]_e$ element strain displacement matrix, see equation (3.2.20)

$[T]_e$ topology matrix describing the relationship between the element displacement vector and the system displacement vector.

$[R]_e$ element rotation matrix transforming local displacement coordinates to global coordinate values.

$\{e\}_i$ unit vector with element 1 in row i

Substituting (3.2.26) and (3.2.27) into the integral on the left hand side of equation (3.2.24) yields:

$$\sum_e \int_{F_e} \frac{h^3}{12} \delta\{\dot{\varepsilon}\}^T [C] \{\dot{\varepsilon}\} df = \sum_e \{e\}_i^T [T]_e^T [R]_e^T \left[\int_{F_e} \frac{h^3}{12} [B]_e^T [C] [B]_e \, df \right] [R]_e [T]_e \{u\}_s \tag{3.2.28}$$

Terms not included under the integral above are independent of x_1 and x_2. The integral term, which is computed numerically, is called the element stiffness matrix $[k]_e$:

$$[k]_e = \int_{F_e} \frac{h^3}{12} [B]_e^T [C] [B]_e \, df \tag{3.2.29}$$

System stiffness matrix: The substitution of (3.2.28) and (3.2.29) into the left side of equation (3.2.24) yields:

$$\sum_e \int_{F_e} \frac{h^3}{12} \delta\{\dot{\varepsilon}\}^T [C] \{\dot{\varepsilon}\} df = \{e\}_i^T \left(\sum_e [T]_e^T [R]_e^T [k]_e [R]_e [T]_e \right) \{u\}_s$$

The term inside the brackets ( ) on the right hand side depends only on the geometry and the material properties of the plate, not on the loading or the support thereof. It is called the system stiffness matrix of the finite element model:

$$[K]_s = \sum_e [T]_e^T [R]_e^T [k]_e [R]_e [T]_e \tag{3.2.30}$$

Element load vector: The first term on the right hand side of equation (3.2.24) is considered. Expressions for the system displacement state and the Galerkin variation with respect to the nodal variable u_{si} are developed:

$$u_3 = \sum_e [S]_e [R]_e [T]_e \{u\}_s \tag{3.2.31}$$

$$\delta u_{3i} = \sum_e [S]_e [R]_e [T]_e \{e\}_i \qquad i \in \{1,\ldots,n\} \tag{3.2.32}$$

in which

$[S]_e$ element displacement interpolation matrix, see (3.2.13)

It is assumed that the surface load varies linearly over the area F_e of a triangular element:

$$p_f = p_{f(1)}z_1 + p_{f(2)}z_2 + p_{f(3)}z_3 = \{p_f\}^T\{z\} \tag{3.2.33}$$

$p_{f(i)}$ surface load intensity at node i

The interpolation function (3.2.32) for the variation of u_3 with respect to u_{si}, and the interpolation function (3.2.33) for the surface load, are substituted into the first integral on the right hand side of (3.2.24):

$$\sum_e \int_{F_e} \delta u_3 p_f \, df = \sum_e \{e\}_i^T [T]_e^T [R]_e^T \left[\int_{F_e} [S]_e^T \left(\{p_f\}^T \{z\} \right) df \right] \tag{3.2.34}$$

Terms not included under the integral term on the right hand side are independent of x_1 and x_2. The integral is computed numerically, and is called the element load vector $\{q\}_e$:

$$\{q\}_e = \int_{F_e} [S]_e^T \left(\{p_f\}^T \{z\} \right) df \tag{3.2.35}$$

System load vector: Substitution of (3.2.32) and (3.2.35) into the first integral on the right hand side of equation (3.2.24) yields:

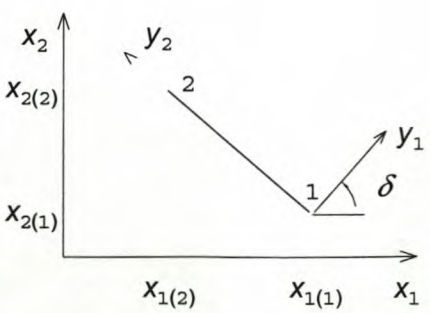
$$\sum_e \int_{F_e} \delta u_3 p_f \, df = \{e\}_i^T \left(\sum_e [T]_e^T [R]_e^T \{q\}_e \right) \tag{3.2.36}$$

The term inside the brackets ( ) on the right hand side above contains only known variables. It is called the system load vector $\{q\}_a$ of the finite element model:

$$\{q\}_a = \sum_e [T]_e^T [R]_e^T \{q\}_e \tag{3.2.37}$$

3.2.3.3 Contribution of the line elements

Line elements: The edge of the plate is composed of line elements. The line elements are used to interpolate the displacements and the stress resultants on the edge of the plate. For the interpolation, a local coordinate system y_1, y_2 of the line element is chosen so that the axis y_1 points in the direction of the outer normal of the edge along the line element, and the axis y_3 points in the direction of axis x_3.



The position of a point on the edge is given by:

$$\{x\} = \{x\}_1 + [R]_\delta \{y\} \tag{3.2.38}$$

$$\begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} x_{1(1)} \\ x_{2(1)} \end{Bmatrix} + \begin{bmatrix} \cos\delta & -\sin\delta \\ \sin\delta & \cos\delta \end{bmatrix} \begin{Bmatrix} y_1 \\ y_2 \end{Bmatrix}$$

Inside the line element, displacements $\{w\}$ are interpolated using (3.2.22). The displacement for the edge as a whole equals the sum of the interpolation functions of the elements. Expressed in terms of the system displacement vector, the edge displacement is:

$$\{w\} = \sum_r [S]_r [R]_r [T]_r \{u\}_s \tag{3.2.39}$$

in which

$[S]_r$ the edge displacement interpolation matrix, see (3.2.22)

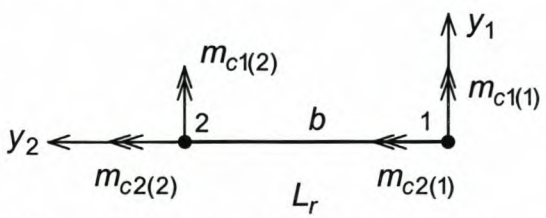
$[R]_r$ the line element rotation matrix which transforms displacement coordinates to the line element coordinate system.

$[T]_r$ line element topology matrix describing the relationship between the line element displacement vector and the system displacement vector.

The i^{th} Galerkin variation of the system edge displacement is:

$$\delta\{w\}_i = \sum_r [S]_r [R]_r [T]_r \{e\}_i \qquad i \in \{1,\ldots,n\} \tag{3.2.40}$$

Concentrated nodal forces and moments: The contribution of line element L_r to the integral equations (3.2.24) is replaced by the contribution of concentrated shear forces $q_{c3(i)}$, and concentrated moments $m_{c1(i)}$, $m_{c2(i)}$ acting at the nodes of L_r. These concentrated forces and moments are assembled into a generalised concentrated nodal force vector $\{q\}_{cr}^T = \{q_{c3(1)} \vdots m_{c1(1)} \vdots m_{c2(1)} \mid q_{c3(2)} \vdots m_{c1(2)} \vdots m_{c2(2)}\}$.



The stress resultants $\{q\}$ are interpolated in the line element with equation (3.2.23). Substitution of (3.2.23) and (3.2.40) into a line integral term of the right hand side of (3.2.24) yields:

$$\sum_r \int_{L_r} \delta\{w\}_i^T \{q\}\, dr = \sum_r \{e\}_i^T [T]_r^T [R]_r^T \left(\int_0^1 [S]_r^T [S]_q\, b\, dz \right) \{q\}_r$$

$$= \sum_r \{e\}_i^T [T]_r^T [R]_r^T \{q\}_{cr} \tag{3.2.41}$$

The matrix relating the nodal stress resultants $\{q\}_r$ to the element force vector $\{q\}_{cr}$ is denoted by $[Q]_r$. Equation (3.2.41) yields:

$$\{q\}_{cr} = [Q]_r \{q\}_r \tag{3.2.42}$$

$$[Q]_r = \int_0^1 [S]_r^T [S]_q\, b\, dz \tag{3.2.43}$$

The integral (3.2.43) is determined analytically.

The contribution of the edge to the integral equation (3.2.23) follows by substitution of (3.2.42) into (3.2.43):

$$\sum_r \int_{L_r} \delta\{w\}^T \{q\}\, dr = \sum_r \{e\}_i^T [T]_r^T [R]_r^T [Q]_r \{q\}_r = \sum_r \{e\}_i^T \{q\}_\ell \tag{3.2.44}$$

The stress resultant vector $\{q\}_\ell$ in (3.2.44) contains both prescribed loads and unknown reactions. When prescribed loads are substituted into (3.2.44), the result is a load vector which is added to the system load vector $\{q\}_s$. The unknown reactions are contained in the system reaction vector $\{r\}_s$.

Nodal loads: Concentrated shear forces and moments can also be applied as external loads, directly at the nodes of the network. It is assumed that the coordinates of these loads are referred to the rotated basis system of the node, if it is defined, and otherwise to the global coordinate system. The prescribed nodal loads are therefore added directly to the system load vector $\{q\}_s$.

3.2.3.4 Algebraic governing equations

The contributions (3.2.28) and (3.2.34) of the triangular elements, the contribution (3.2.44) of the line elements, and the contribution of the directly applied node loads are substituted into the governing equation (3.2.24):

$$\{e\}_i^T [K]_s \{u\}_s = \{e\}_i^T \left(\{q\}_a + \{q\}_\ell + \{q\}_k + \{r\}_s\right) \qquad i \in \{1,\ldots,n\}$$

Each value of i contributes one algebraic governing equation. The total of n equations is written in matrix form:

$$[K]_s \{u\}_s = \{q\}_s + \{r\}_s \tag{3.2.45}$$

$[K]_s$ system stiffness matrix (3.2.30)

$\{u\}_s$ system displacement vector (3.2.25)

$\{q\}_s$ system load vector (3.2.25), equals $\{q\}_a + \{q\}_\ell + \{q\}_k$

$\{q\}_a$ system area load vector (3.2.37)

$\{q\}_\ell$ system line load vector (3.2.44)

$\{q\}_k$ system node load vector

$\{r\}_s$ system reaction vector

3.2.4 Results

System displacement and reaction vectors: Equation (3.2.45) is a system of linear equations which can be solved for an appropriate set of prescribed nodal displacements and loading[1]. The coordinates of prescribed displacements are referred to the rotated basis system at a node, if it is defined, otherwise to the global coordinate system. If the subscripts u and p respectively refer to unknown and prescribed coordinates of the system displacement vector, (3.2.45) can be written in the form:

$$\begin{bmatrix} [K]_{uu} & [K]_{up} \\ [K]_{pu} & [K]_{pp} \end{bmatrix}_s \begin{Bmatrix} \{u\}_u \\ \{u\}_p \end{Bmatrix}_s = \begin{Bmatrix} \{q\}_u \\ \{q\}_p \end{Bmatrix}_s + \begin{Bmatrix} \{0\} \\ \{r\}_p \end{Bmatrix}_s \tag{3.2.46}$$

[1] Aspects of the solution process are described in Chapter 4

From the first equation in (3.2.46), the unknown system displacement coordinates $\{u\}_u$ are computed by solving:

$$[K]_{uu}\{u\}_u = \{q\}_u - [K]_{up}\{u\}_p \tag{3.2.47}$$

Once $\{u\}_u$ has been computed, the unknown reaction forces $\{r\}_p$ follow from the second equation in (3.2.46):

$$[K]_{pu}\{u\}_u + [K]_{pp}\{u\}_p = \{q\}_p + \{r\}_p \tag{3.2.48}$$

Element displacements, strains, stresses and stress resultants: Once the system displacement coordinates are known, element nodal displacements are extracted from the system vector using the element topology matrix. The transverse displacement inside the element is interpolated using (3.2.13), and the strains follow from (3.2.20). Stress coordinates are computed using (3.1.5) after which the bending moment vectors $\{m\}_1$ and $\{m\}_2$ are obtained using (3.1.10) and (3.1.11). The shear stress resultants, equations (3.1.15) and (3.1.16), require the 3rd partial derivatives of the transverse displacement:

$$q_{13} = -\frac{Eh^3}{12\left(1-\nu^2\right)}\frac{\partial}{\partial x_1}\left(u_{3,11} + u_{3,22}\right) \tag{3.1.15'}$$

$$q_{23} = -\frac{Eh^3}{12\left(1-\nu^2\right)}\frac{\partial}{\partial x_2}\left(u_{3,11} + u_{3,22}\right) \tag{3.1.16'}$$

The global 3rd derivatives of an arbitrary element f_i of the form vector $\{f\}$ are determined with the chain rule:

$$\frac{\partial^3 f_i}{\partial x_j \partial x_k \partial x_m} = \sum_r \sum_s \sum_t \frac{\partial^3 f_i}{\partial z_r \partial z_s \partial z_t}\frac{\partial z_r}{\partial x_j}\frac{\partial z_s}{\partial x_k}\frac{\partial z_t}{\partial x_m}$$

The normalised 3rd derivatives of the form vector, $\partial^2\{f\}/\partial z_r \partial z_s \partial z_t$, are readily determined from equation (3.2.14). The global derivatives of the normalised coordinates, $\partial\{z\}/\partial x_k$, are given by equation (3.2.6). For the special case of the linear triangle, the form matrix $[P]_e$ is constant inside the element. The global 3rd derivative of u_3 follows from (3.2.13):

$$\frac{\partial^3 u_3}{\partial x_j \partial x_k \partial x_m} = \frac{\partial^3 \{f\}^T}{\partial x_j \partial x_k \partial x_m}[P]_e\{u\}_e \tag{3.2.49}$$

Substitution of (3.2.48) into (3.1.15) and (3.1.16) yields the shear stress resultants.

4 Object oriented models for structural analysis

The algorithmic basis for the numerical analysis of structures is described in general in chapter 2, and in detail, for thin plate analysis, in chapter 3. The method consists of developing governing equations for the structural components and finite element theory for the solution of the equations. A model with the necessary components and functionality to execute the finite element algorithms has to be implemented on the computer to analyse a given structure.

In this chapter the implementation of an object oriented model for finite element analysis in local environments is described. As an introduction, mathematical background relating to the structuring of models and the ordering of objects is presented in section 4.1. The finite element model is described in sections 4.2 through 4.6. In section 4.2 the scope of the implementation is described.

Structuring an application around the use of persistent identifiers to manage the relations between objects is described in section 4.3. It turns out that employment of this technique makes a crucial contribution to reducing of complexity of management in the distributed environment. As such it forms the basis of methods developed for distributed analysis collaboratories, as described in later chapters.

The implementation of the model components is described in section 4.4, and the implementation of the finite element model is presented in section 4.5. The class diagram shown in section 4.6 gives a concise overview of the implementation.

In a distributed structural analysis collaboratory, finite element analyses have to be performed in the distributed communication network. The basic implementation concepts described in this chapter have to be extended and adapted to deal with the requirements of this scenario. The scope of models for distributed collaboratories is developed in the next chapter.

4.1 Structuring of models and ordering of objects

A model maps mental concepts regarding a system of the real world, to the computer. The object oriented approach is based on the premise that real world systems, built facilities are under investigation in the dissertation, consist of separable entities which have certain properties and functionalities. The entities are mapped to a structured set of software objects called an object model. The structure of the object model is laid down in the relations on the objects. The class structure of the object model is a central concept in object orientation, and is the mechanism for implementing equivalence relations in object models. Furthermore, mapping and ordering relations play an essential role in datastructures and the ordering of objects. In order to clarify these perspectives, some mathematical essentials regarding

relations in general and the mentioned relations in particular are presented below. This is followed by a description of datastructuring and ordering of objects in sequences.

Relation: Consider two sets A and B. The set of all ordered pairs (a,b), formed by pairing the elements $a \in A$ and $b \in B$, is called the cartesian product of the sets A and B:

$$\text{Cartesian product:} \quad A \times B := \{(a,b) \mid a \in A \wedge b \in B\}$$

If a logical rule is introduced to govern the selection of ordered pairs from the cartesian product, the value (true or false) of the combination for the ordered pair $(a,b) \in A \times B$ is denoted by aRb. The subset of the cartesian product $A \times B$ for which aRb is true, is called a relation in A and B, and is denoted by the symbol R. A relation in A and B is the set:

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

Properties of a relation: When a relation is formed in a cartesian product $M \times M$, then $R \subseteq M \times M$ is called a relation in M. The following properties of relations are defined:

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

R is reflexive $\quad :\Leftrightarrow \underset{a}{\wedge}(aRa)$

R is anti-reflexive $\quad :\Leftrightarrow \underset{a}{\wedge}(\neg aRa)$

R is symmetric $\quad :\Leftrightarrow \underset{a\ b}{\wedge\wedge}(aRb \Rightarrow bRa)$

R is anti-symmetric $\quad :\Leftrightarrow \underset{a\ b}{\wedge\wedge}(aRb \Rightarrow \neg bRa)$

R is identitive $\quad :\Leftrightarrow \underset{a\ b}{\wedge\wedge}(aRb \wedge bRa \Rightarrow a = b)$

R is transitive $\quad :\Leftrightarrow \underset{a\ b\ c}{\wedge\wedge\wedge}(aRb \wedge bRc \Rightarrow aRc)$

R is linear $\quad :\Leftrightarrow \underset{a\ b}{\wedge\wedge}(aRb \vee bRa)$

R is connex $\quad :\Leftrightarrow \underset{a\ b}{\wedge\wedge}(a \neq b \Rightarrow aRb \vee bRa)$

4.1.1 Equivalence relation

Equivalence relation: A relation $E \subseteq M \times M$ is called an equivalence relation in the set M when it is reflexive, symmetric and transitive. Elements x and y of M are equivalent when the set E contains the ordered pair (x,y), and due to symmetry also the pair (y,x). The equivalent pair is denoted by $x \sim y$.

$x \sim x$ (reflexive)

$x \sim y \Rightarrow y \sim x$ (symmetric)

$x \sim y \wedge y \sim z \Rightarrow x \sim z$ (transitive)

Equivalence class: A subset, containing all pairwise equivalent elements in a set M, is called an equivalence class in M. An equivalence class is denoted by an arbitrary element, say r, of the class. It is called the representative of the equivalence class:

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

Decomposition into equivalence classes: For a given equivalence relation $E \subseteq M \times M$, it follows from the properties of the equivalence relation:

- that each element $x \in M$ is an element of an equivalence class, since (x,x) is an element of the reflexive relation E,
- that none of the equivalence classes $[x]$ is empty, since $(x,x) \in E$ and at least x itself is an element of $[x]$,
- that each element $z \in M$ is an element of exactly one equivalence class. If z were an element of the classes $[x]$ and $[y]$, then $z \sim x$ and $z \sim y$. Since E is symmetric, $x \sim z$. Since E is transitive, $x \sim z \wedge z \sim y \Rightarrow x \sim y$, thus $[x] = [y]$.

Consequently, for a given equivalence relation E, the equivalence classes in the set M are a decomposition of M. It is important to note that "concepts" are actually equivalence relations. Concepts, e.g. "corrodability" (of metal components), or "axial loadbearing capacity" (of steel profiles), or "parallelism" (of lines), are reflexive, symmetric and transitive relations which lead to decompositions of given sets.

Classes and objects: The equivalence relation provides the mathematical structure on which the class concept of object oriented models is based. The objects in a model are classified, each class representing a set of equivalent objects (i.e. an equivalence class) which is a subset of the set comprising the model. The class (as concept) is an abstraction of equivalent objects, i.e. it is a datatype for objects with equivalent properties and behaviour. The properties are specified by attribute variables, and the behaviour by the methods of the class. Each object is an instance of a class. It knows its class, has its own data for its attribute variables, and shares the methods determining its behaviour with other objects of the class. Information can also be stored in static variables, or class variables. This information is not encapsulated in an instance of a class (object), but is stored with the class. Similarly, methods belonging to a class, but which are not bound to instances of the class, are called class methods or static methods.

4.1.2 Ordering relation

Ordering of objects: Structural design of built facilities constantly requires the identification of behaviour variables whose value exceeds a certain magnitude, or whose values fall within a given range, etc. This information is carried in objects of the finite element model, and to efficiently identify relevant objects in such enquiries the first requirement is that the set containing the objects should be ordered. Ordering of objects in a set is accomplished by establishing an order relation in the set. Ordering relations are defined below.

Ordering relation: A relation in a given set M is an ordering relation when it is a reflexive, identitive and transitive relation. Ordering relations are denoted by the symbols $\sqsubseteq$ or $\leq$. An ordering relation $\sqsubseteq$ in the set M is then, per definition, a subset of the cartesian product $M \times M$ with the following properties for the elements $a, b, c \in M$:

(1) $\sqsubseteq$ is reflexive : $a \in M \quad \Rightarrow \quad a \sqsubseteq a$

(2) $\sqsubseteq$ is identitive: $a \sqsubseteq b \wedge b \sqsubseteq a \quad \Rightarrow \quad a = b$

(3) $\sqsubseteq$ is transitive: $a \sqsubseteq b \wedge b \sqsubseteq c \quad \Rightarrow \quad a \sqsubseteq c$

Partially ordered set: In the definition of the ordering relation above, comparability of all elements of the set is not a requirement. The structure $(M; \sqsubseteq)$ is a partially ordered set, i.e. the set M is partially ordered by the relation $\sqsubseteq$.

Comparable elements: Two elements a, b of a partially ordered structure $(M; \sqsubseteq)$ are comparable when either (a,b) or (b,a) are elements of the relation $\sqsubseteq$, i.e. either $a \sqsubseteq b$ or $b \sqsubseteq a$. In general not each element of a set is comparable to all the other elements of the set.

Total ordering relation: An ordering relation $\sqsubseteq$ in a set M is total when any two elements of M are comparable by the ordering relation. In addition to the requirements (1) through (3), a total ordering relation is also linear:

(4) $\sqsubseteq$ is linear : $a, b \in M \quad \Rightarrow \quad a \sqsubseteq b \ \vee \ b \sqsubseteq a$

Totally ordered set: A partially ordered structure $(M; \sqsubseteq)$ is totally ordered when the ordering relation $\sqsubseteq$ is total. Any two elements of a totally ordered set are comparable by the ordering relation.

Strict ordering relation: A relation in a given set M is a strict ordering relation when it is anti-reflexive, anti-symmetric and transitive. Strict ordering relations are denoted by the symbols $\sqsubset$ and $<$. A strict ordering relation $\sqsubset$ in the set M is then, per definition, a subset of the cartesian product $M \times M$ with the following properties for the elements $a, b, c \in M$:

(1) $\sqsubset$ is anti-reflexive: $a \in M \quad \Rightarrow \quad \neg (a \sqsubset a)$

(2) $\sqsubset$ is anti-symmetric: $a \sqsubset b \quad \Rightarrow \quad \neg (b \sqsubset a)$

(3) $\sqsubset$ is transitive: $a \sqsubset b \wedge b \sqsubset c \quad \Rightarrow \quad a \sqsubset c$

Partially strictly ordered set: In the definition of the strict ordering relation above, comparability of elements of the set is not a requirement. Given a set M and a strict ordering relation $\sqsubset$, then the structure $(M; \sqsubset)$ is a partially strictly ordered set. The set M is partially strictly ordered by the relation $\sqsubset$.

Comparable elements: Two elements $a \neq b$ of a partially strictly ordered structure $(M; \sqsubset)$ are comparable when either (a,b) or (b,a) are elements of the relation $\sqsubset$, i.e. either $a \sqsubset b$ or $b \sqsubset a$.

Total and strict ordering relation: A strict ordering relation $\sqsubset$ in a set M is total when any two elements of the set are comparable by the ordering relation. In addition to the requirements (1) through (3), a total and strict ordering relation is also connex:

(4) $\sqsubset$ is connex : $a,b \in M \quad \Rightarrow \quad (a \neq b \quad \Rightarrow \quad a \sqsubset b \ \vee \ b \sqsubset a)$

Totally strictly ordered set: A partially strictly ordered structure $(M; \sqsubset)$ is strictly ordered when the strict ordering relation $\sqsubset$ is total. Any two elements of a totally strictly ordered set are comparable.

4.1.3 Mapping relation

A relation $R \subseteq A \times B$ in general does not constitute a unique association between a specific element $a \in A$ and a specific element $b \in B$. For elements $c, x \in B$ both the pairs (a,c) and (a,x) may be elements of the relation R. A relation for which each element of A is paired with exactly one element of B is called a map. Mapping is used in many applications, and is an essential concept in datastructuring, as explained in section 3.1.4. Datastructuring, in turn, is required to realise an order relation in a set of objects in computer memory. The properties of mapping relations are defined below.

Completeness of a relation: A relation $R \subseteq A \times B$ is left-complete if, for each $a \in A$, there is at least one $b \in B$ such that (a,b) is an element of R. The relation R is right-complete if, for each $b \in B$, there is at least one $a \in A$ such that $(a,b) \in R$. A relation which is left and right-complete is a complete relation.

R is left-complete : $\quad :\Leftrightarrow \quad \underset{a}{\wedge} \underset{b}{\vee} (aRb)$

R is right-complete: $\quad :\Leftrightarrow \quad \underset{b}{\wedge} \underset{a}{\vee} (aRb)$

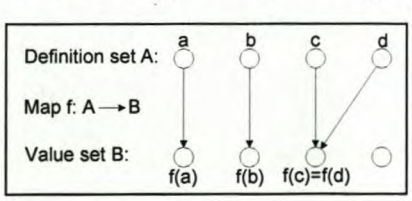
Uniquenes of a relation: A relation $R \subseteq A \times B$ is left-unique if, for each $a \in A$, there is exactly one $b \in B$ such that (a,b) is an element of R. The relation R is right-unique if, for each $b \in B$, there is exactly one $a \in A$ such that $(a,b) \in R$. A relation which is left and right-unique is a unique relation.

R is left-unique : $\quad :\Leftrightarrow \quad \underset{a}{\wedge} \underset{b}{\wedge} \underset{c}{\wedge} (aRb \wedge cRb \ \Rightarrow \ a = c)$

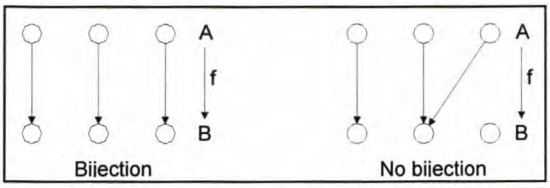
R is right-unique : $\quad :\Leftrightarrow \quad \underset{a}{\wedge} \underset{b}{\wedge} \underset{c}{\wedge} (aRb \wedge aRc \ \Rightarrow \ b = c)$

Mapping: A relation $f \subseteq A \times B$ is a map if it is left-complete and right-unique. Mapping relations are usually expressed as:

$f : A \rightarrow B$	f is a map of A to B
A	Definition set of f
B	Value set of f



Bijective map: A map $f : A \rightarrow B$ is bijective (i.e. the map is a bijection) if each element of B is the mapped element of exactly one element of A. A bijection is a complete, unique relation. The number of elements in A is equal to the number of elements in B.



Inverse map: For each bijective map $f : A \rightarrow B$ there exists an inverse map $f^{-1} : B \rightarrow A$. From $f(a) = b$ follows $f^{-1}(b) = a$. If the map f is not bijective it does not have an inverse.

4.1.4 Datastructure and ordering

Mathematical structure: Consider a given set of objects which are to be stored in computer memory, or read from computer memory. A general set is per definition unstructured, it contains no information concerning its sequence or position in computer memory. Relations are employed to mathematically structure the set. Equivalence relations are used to classify the set, as explained above. Ordering relations are used to partially or totally order the set.

Storage structure: The elements of a set in the computer are objects. The positioning of these objects in the bytecode is called the storage structure. Each element has two storage fields, namely the reference and the body of the object. These storage fields are in general not sequentially laid down in the bytecode. The relative positioning of the bodies of different objects cannot be influenced by the programmer, it is assigned by the Java runtime environment. The programmer can, however, determine the relative positioning of these objects' references, e.g. by sequencing them in an array or in the body of an object.

Datastructure: The modeling of a set in the computer requires the following resolutions:

- The mathematical structure of the set, as basis of the modeling, is chosen. The elements are, for example, positioned in a sequence in which their position is fixed.
- The storage structure for the relative positioning of the objects' references is chosen. The references are, for example, sequentially stored in an array.
- The mapping rule, according to which the mathematical structure of the set is bijectively mapped to the storage structure, is chosen.

A chosen mathematical structure, a chosen storage structure, and the mapping rule between these two comprise a datastructure for the set. A set may have different datastructures for different applications.

In order to demonstrate these concepts, mathematical structures for ordering objects into unsorted and sorted sequences are described below. Two possible storage structures are also pointed out.

4.1.4.1 Unsorted sequence

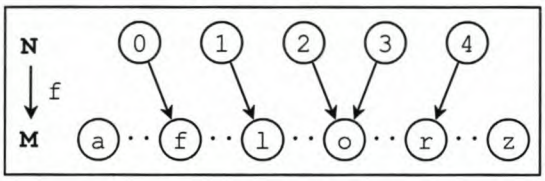
Mathematical structure: The first step in ordering a set M is to arrange its elements into a fixed sequence. This can be done by mapping a totally ordered set, e.g. the natural numbers $0,1,\ldots,n-1$, to the set. This map is denoted as a sequence.

Unsorted sequence: Given an unstructured set M and a subset N of the natural numbers, the map of N to M is denoted as a sequence. Then, for $i \in N$ mapped to $a_i \in M$:

$$f \text{ is a sequence} \quad :\Leftrightarrow \quad f : N \to M \quad \text{with } f(i) = a_i$$

By definition of a map as a left-complete, right-unique relation, the sequence f is a set of ordered pairs (i, a_i), with each $i \in N$ having a mapped element in M, and i's mapped element $a_i \in M$ is unique.

Example: Given the set of small letters $M := \{a,b,c,\ldots,z\}$. The word "floor" is a map of the subset $N := \{0,1,2,3,4\}$ of the natural numbers on the set M:



Sequence: $f : N \to M$

$f := \{(0,f), (1,l), (2,o), (3,o), (4,r)\}$

The sequence can also be denoted as a vector:

0	1	2	3	4
f	l	o	o	r

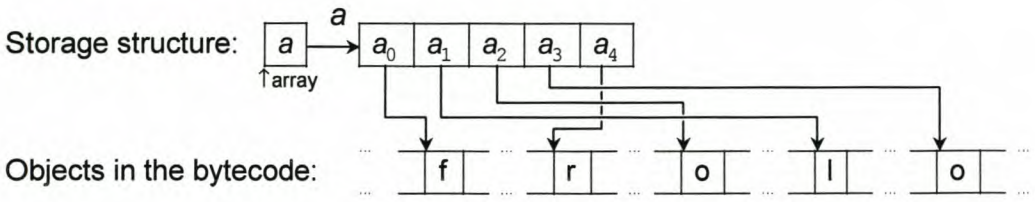
or as a row: <f,l,o,o,r>

Storage structure for sequences: The references of objects in a sequence may be stored either sequentially in the bytecode, or in a distributed way. If the references of a sorted sequence is stored sequentially, a specific element can be found without much effort. However, inserting or deleting elements in a sequentially stored sequence requires much effort, since all references following the point of change have to be moved to new storage positions. This problem can be avoided by distributed storage of the references in a linked list (chain). Adding or removing a link from a linked list only impacts on the neighbouring links, and is consequently independent of the length of the linked list. Searching for a specific element of the sequence in a linked list is, however, more expensive than searching in sequential storage since each link, from the root of the linked list, has to be queried until the target is found.

Sequential storage structure: The order of elements in a sequence $< a_0, a_1, ..., a_{n-1} >$ is mapped in the computer by storing the references $a_0, a_1, ..., a_{n-1}$ of the objects in the sequence in consecutive bytes of the bytecode:

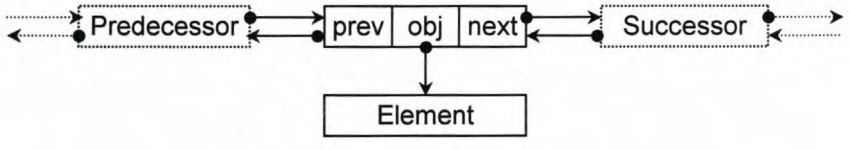
Mathematical structure:

0	1	2	3	4
f	l	o	o	r



In Java the class ArrayList of the package java.util provides a datastructure for sequential storage of sequences (lists).

Distributed storage structure: A linked list serves as a distributed storage structure for a sequence. Each element of the linked list is a link object. The first link is called the root. The links of a linked list are connected by references. In a singly linked list, each link has the reference of its successor (i.e. the next link in the list). In a doubly linked list each link has the references of both its predecessor and its successor. A typical link in a doubly linked list is shown.



prev: Reference of predecessor.
next: Reference of successor.
Element: Reference of the element in the sequence.

In Java the class LinkedList of the package java.util provides a datastructure for the distributed storage of sequences (lists).

4.1.4.2 Sorted sequence

Mathematical structure: Consider a given set of elements M. The elements are per definition similar in the sense that each of the elements can be assigned a value from a given set V of possible attribute values. If V is, e.g., a set of masses per unit length of steel profiles, then each element of M has the attribute "mass per unit length". Such an attribute is called a property of the element. The value of an element's property is assigned by the map:

$g : M \rightarrow V$ with $g(a) = v_a$

M Set of elements

V Set of possible attribute values

The map g is a set of ordered pairs (a,v_a). Different elements $a,b \in M$ may have the same mapped value, i.e. $v_a = v_b$. However, the ordered pairs (a,v_a) and (b,v_b) are not equal since $a \neq b$. The ordered pairs (a,v_a) are consequently unique.

$$g = \{(a,v_a) \mid a \in M\}$$

g Property set

Ordering relations for property sets: For the attribute set V an ordering relation $<$ is given. Then the structure $(V;<)$ is ordered. An ordering relation $\sqsubset$ for the property set g is associated with the ordering relation $<$ of the attribute set V. The pair (a,v_a) is "less than" the pair (b,v_b) if $v_a < v_b$ is true.

$$(a,v_a) \sqsubset (b,v_b) \;:=\; v_a < v_b$$

$(g;\sqsubset)$ ordered property set

Property sequences: For a set M, an element sequence $f : N \to M$ with $N = \{0,1,\ldots,n-1\}$ and $f(i) = a_i$ is defined. Each element of M has a property v. The value of v for each element of M is given by the map $g : M \to V$ with $g(a_i) = v_i$. Then the composition $h = g \circ f$ of the maps f and g constitutes a property sequence:

$$f : N \to M \;\wedge\; g : M \to V \quad \Rightarrow \quad h : N \to V$$

N	0	1	2	3
$f\downarrow$				
M	a_0	a_1	a_2	a_3
$g\downarrow$				
V	v_0	v_1	v_2	v_3

$h = <(0, v_0),\ldots,(n-1, v_{n-1})>$

h Property sequence

Partially sorted sequence: The element sequence $f : N \to M$ is partially sorted according to the property $g : M \to V$ if, in the property sequence $h = g \circ f$ the implication $i < m \;\Rightarrow\; \neg(v_m < v_i)$ is true for all $i,m \in N$. If the position of a_i is to the left of that of a_m in the sequence f, then, in the sorted sequence, the attribute v_m is not less than the attribute v_i. This requirement is also considered to be satisfied when v_i and v_m are not comparable.

$$f \text{ is partially sorted} \;:\Leftrightarrow\; \bigwedge_{i \in N} \bigwedge_{m \in N} (i < m \;\Rightarrow\; \neg(v_m < v_i))$$

Totally sorted sequence: The element sequence $f : N \to M$ is totally sorted according to the property $g : M \to V$ if, in the property sequence $h = g \circ f$, the equivalence $i < m \;\Leftrightarrow\; v_i < v_m$ is true for all $i,m \in N$. If the position of a_i is to the left of that of a_m in the

sequence f, then, in the totally sorted sequence, the attribute v_i is less than the attribute v_m.

$$f \text{ is totally sorted } :\Leftrightarrow \bigwedge_{i \in N} \bigwedge_{m \in N} (i < m \Leftrightarrow v_i < v_m)$$

This criterium for a sequence to be totally sorted is also satisfied if the criterium for the sequence to be partially sorted is satisfied. Consequently, algorithms for sorting are developed for the criterium for sequences to be partially sorted.

Unique sequence: A sequence is unique if only one sequence $f : N \rightarrow M$ satisfies the sorting criterium. Whether a sequence is unique is determined by the properties of the ordering relation and the attribute values in the following ways:

- If the ordering relation is not strict, the rules for strict ordering relations still apply if no two attribute values v_i, v_m are equal.
- If the ordering relation is not total, the rules for total ordering relations still apply if each pair $\{v_i, v_m\}$ of the actual attribute values of the elements is comparable.
- If the ordering relation is strict and total, the sequence is unique. For random indices $i \neq m$, either $v_i \sqsubset v_m$ or $v_m \sqsubset v_i$. This determines the relative positions for each element pair $\{a_i, a_m\}$ in the sequence.
- If the ordering relation is not strict the sequence is not unique: For the case $v_i = v_m$, switching the elements a_i and a_m in the sequence does not violate the sorting criterium.
- If the ordering relation is not total the sequence is not unique: If two elements a and b are not comparable, both may have the same successor x in the ordering diagram, i.e. $v_a \sqsubset v_x$ and $v_b \sqsubset v_x$. Switching the elments a and b in the sub-sequence $< a, b, x >$ to $< b, a, x >$ does not violate the ordering criterium.

4.1.4.3 Sorting and searching

Mathematical background pertaining to ordering of objects into sequences, and of sorting such sequences is described above. Sorting algorithims which implement those concepts are employed to establish partially or totally sorted sequences of objects, and search algorithms are implemented to efficiently find specific information in appropriately sorted sequences. An extensive range of sorting and searching algorithms for linear data structures (sequences) is described in the literature [Li 1998] [Knuth], and have been implemented in software as well.

Java supports sorting of totally ordered sets only, i.e. any two objects of a set which is to be sorted must be comparable. The ordering relations are defined by implementing the Java interfaces Comparable (of package java.lang) and Comparator (of package java.util). Sorting and searching methods of the classes Arrays and Collections of the package java.util can be called to perform the sorting and searching tasks in datastructures whose elements are comparable.

4.1.4.4 Datastructures and Java implementation

Functionality and efficiency: Linear datastructures, which organize elements into a sequence as described above, represent only one possibility out of an extensive range of possible datastructures which have been developed. In general, the mathematical structure establishes the functionality, while the storage structure governs the efficiency of a datastructure. A specific task requires specific functionality, and consequently determines the datastructure which is appropriate. Considering, for example, the sequential storage datastructure (array) and distributed storage datastructure (linked list), the latter has the advantage over the array of fast insertion and removal of elements at any point in the sequence. However, searching in a sorted array is much faster than searching in linked list. This problem was solved by the development of tree-structures, which allow fast insertion and removal of elements, while providing for fast searching as well. Another datastructure, the hash table, is an efficient structure for managing (key, value) pairs.

Implementation: The Java language provides extensive support of datastructures in the package java.util of its applications programming interface. The Java interface Collection is the fundamental interface for sets of objects. The interfaces Set, List extend interface Collection, and interface SortedSet in turn extends interface Set. The fundamental interface for sets of mappings is Map. The interface SortedMap extends interface Map. The interfaces are implemented using different storage structures in classes whose names are the combination of the storage structure employed and the interface which is implemented. Array storage of a sequence (called a list in Java) is provided by class ArrayList. Datastructuring classes are listed in table 4.1 below.

	Java interface				
Storage structure ↓	Set	List	Map	SortedSet	SortedMap
Hash: Mapping	HashSet		HashMap		
Tree: Balanced tree				TreeSet	TreeMap
Array: Contiguous sequential		ArrayList			
Link: Distributed sequential		LinkedList			

Table 4-1: Java data structures

4.2 Scope of the implementation

The finite element theory for thin plates, developed in chapter 3, is implemented in a finite element model for use in a local environment. The model consists of triangular finite elements with three degrees of freedom at each node, i.e. the transverse displacement and rotations about the x_1 and x_2 axes. The model can be used for linear analysis of thin plates with arbitrary shape and boundary conditions.

Platform: The model is implemented in Java, and unless specified otherwise all references to implementation aspects imply usage of the Java language. The overall structure of the implementation, as well as the structure of its classes, methods, and database have been kept as simple as possible. No effort has been made to optimize memory requirements or computation time.

Limitations: The model as implemented does not have the functionality required for engineering practice. A full grapical user interface for data input and display of results is not provided, and error control of user input is rudimentary. No provision has been made for the handling of load cases or load combinations, the determination of extreme values of displacements or stress resultants, different versions of the plate model or for several plate models in one application. Automatic generation of suitable system indices for nodal variables and accuracy control through mesh adaption are not available. Implementation of such utilities involves a large programming effort. For the purposes of the dissertation, those aspects which have a bearing on distributed applications are adressed in following chapters.

Description of an analysis task: For the execution of an analysis task, the user has to specify the geometry and material of the structure and its static and kinematic boundary conditions. This is done by instantiating appropriate component objects of the finite element model. This model is analysed by executing the finite element algorithms, and the user can then study the behaviour. For these purposes a user interface is provided through which a finite element analysis can be instantiated, component objects can be added to the finite element model, the analysis algorithims can be executed, and results can be printed. The data required for instantiation of component objects are entered in a text file, one object per line. The format of a data line for an object of a certain class has to reflect the signature of a constructor of the class. Comment lines are allowed (by entering // as the first two characters) and the objects may be defined in arbitrary order. The component classes, and the information required for instantiation of an object of the class are listed below.

- Node: The identifier of the node, the index of the first nodal degree of freedom in the system vector, and its global coordinates are specified. The other nodal degrees of freedom are assigned consecutive indices. Optionally, a local coordinate system can be associated with the node by specifying the identifier of the local coordinate system.

- Local: The identifier of a local system and the angle from the global axis x_1 to the local axis y_1 are specified. The angle is measured in degrees, and the counterclockwise direction is positive.
- Support: The identifier of the support and the identifier of the node at which the support is located are specified. General supports are specified with a status variable (true/false) and a value for each nodal displacement component. If the status of a component is true, its value is prescribed. Otherwise, the displacement is computed. Simple and fixed supports can be specified with symbolic constants, which are defined as follows:

  SIMPLE $\Rightarrow$ status := {true, false, false}
  value := {0.0, *, *}
  FIXED $\Rightarrow$ status := {true, true, true}
  value := {0.0, 0.0, 0.0}

- Material: The identifier of the material, its modulus of elasticity and its Poisson's ratio are specified.
- Element: The identifier of the element, the identifiers of its three nodes, the identifier of the element material and the element thickness are specified.
- NodeLoad: A concentrated load may only be applied at a node. The identifier of the nodal load, the identifier of the node at which it is applied, and the intensity of the load are specified. If a local coordinate system is specified at the node, the coordinates of the nodal load are referred to the local coordinate system. The load intensity is as follows:

  q_{c3}, m_{c1}, m_{c2} load coordinates

- LineLoad: A distributed load on a line between two adjacent nodes may be specified. The identifier of the line load, the identifiers of its end nodes, and the intensity of the load are specified. If the line load is constant, its components are specified in one of the following forms:

  q_3 load in direction y_3, $m_1 = m_2 := 0$
  q_3, m_1, m_2 load coordinates in the local coordinate system

  If the line load varies linearly, its components are specified in one of the following forms:

  q_{31}, q_{32} load q_3 (in direction y_3) at the nodes, $m_1 = m_2 := 0$
  q_{31}, m_{11}, m_{21} local load coordinates at the start node, and
  q_{32}, m_{12}, m_{22} local load coordinates at the end node

- AreaLoad: The identifier of the distributed load, the identifier of the element to which it is applied, and the intensity of the load are specified. The intensity consists of a single value if the distributed load is constant. If the load varies linearly, the load intensity at the corner nodes of the element are specified in the sequence in which the nodes are listed in the construction of the element.

Printing results: Output of results is in alpha-numeric form and line oriented. Output selection is provided through a tree structure:

- The branch "Model" prints tables of data, as defined by the user, for
  - nodes, and/or
  - local systems, and/or
  - elements and/or
  - materials.
- The branch "Influences" prints tables of data, as defined by the user, for
  - supports, and/or
  - node loads, line loads, and area loads.
- The branch "Behaviour" prints tables of data, as computed, for
  - nodes: the nodal displacements in the global coordinate system
  - elements: moments at the center of the elements in the global coordinate system, or
  - supports: generalised displacements and forces in the local coordinate systems.

4.3 Structure of an application

4.3.1 Objects and classes of an application

Application: An application is the software which is installed on a computer for the treatment of a specific set of tasks. In this implementation, plate analysis is treated as if it were an application. The scope of numerical structural analysis was described in chapter 2, and a complete application would allow for the analysis types and structural components as described there. The development of a complete application requires many person years of effort.

Objects: The description of the scope of the implementation in section 4.2 indicated that the model is composed of objects. The objects are classified as nodes, elements, supports, etc. Each class of object is treated as a generalised data type composed of attributes and methods, as defined in Java. It is assumed that each object belongs to exactly one application. The management of the objects can therefore be restricted to one application. In this application, the management of objects and their associations is based on the use of persistent identifiers.

Persistent identifiers: Each object of the application has a unique identifier. The scope of uniqueness (namespace) of the identifier is the application. This permits the creation of sets containing objects from different classes without ambiguity in their identification. The application uses two identifiers for each object. The external identifier is a String, which is assigned by the user. The internal identifier is a Java reference to the object, which is created at runtime. The external identifier of an object is persistent: it does not vary from one execution of the application to the next. In contrast, the internal identifier depends on the location of the

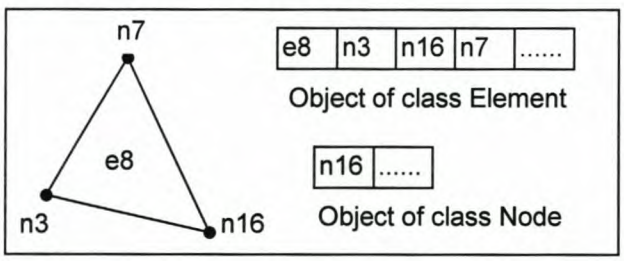
object in storage and is therefore temporary: it varies from one execution of the application to the next.

The differentiation between internal and external identifiers is essential. It permits the specification of an object, which has not yet been constructed, as a property of another object. For instance, the identifier of a node can be specified as a property of an element even though the node object has not yet been constructed. A similar procedure with references is not possible, since the reference of an object only becomes available after the object has been constructed.

For internal processing, the associations between objects must be described with references. The conversion of external identifiers to internal references is achieved in the class App(lication), which is described in section 4.3.3

4.3.2 Relations between objects

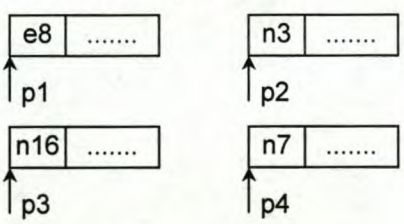
External specification: The user relates objects by their external identifiers. For instance, if an element with identifier "e8" has nodes with the identifiers "n3", "n16", "n7", these node identifiers are stored in the object of class element.



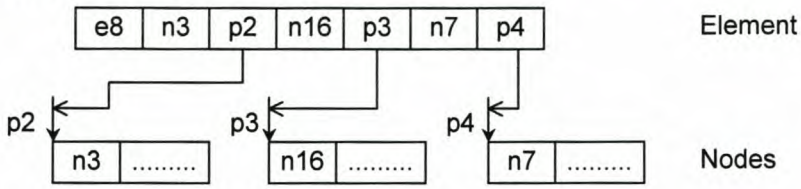
For internal processing, element "e8" must access the attributes of node objects such as "n16". The element "e8" must therefore know the reference of the node object "n16". This is achieved with an object map in class App.

Object map: The object map is a static attribute of class App. The map is of datatype HashMap, which is defined in the package java.util. Every object of the application is entered in the object map when it is constructed. The entry consists of the String identifier as key, and the reference identifier as value. All objects for the input of the model are entered in the object map before the internal relationship between the objects is established. For the example above, the object map would contain the following entries:

key	value
e8	p1
n3	p2
n16	p3
n7	p4



Connection: After the input of the model is completed, and before its behaviour is analysed, its objects are connected by determining the internal references which correspond to the external identifiers. In the case of the object "e8", the String identifiers which are stored in the object are used as keys in the object map to determine the value of the references p2, p3, p4 of the node objects. These are then stored in object "e8" and permit the element object to access the node objects.



Classes of the application that have associations with other classes implement the object method setReferences(). The task of this method is to establish the connections as explained above. In line with the example above, objects of class Element activate their associations with Node objects as shown in the code below.

```
public class Element extends AppObject implements IcElement{
...
String     nodeId[]    ;       // node identifiers (anticlock)
IcNode     node[]      ;       // node references
...
public void setReferences() throws FemException{
  for (int i = 0; i < 3; i++){                  // Set node references
    node[i] = (Node)App.getObject(nodeId[i]) ;
    if (node[i] == null) throw new FemException
    ("*** Node " +nodeId[i] + " of Element " + id + " is not defined
***");}
  ...                                           // Set other references
}
...
}
```

4.3.3 Class App(lication)

Scope: The HashMap objectMap of this class contains the identifiers and the references of all objects of the application. The static methods of class App are used by other classes to enter, delete and count the objects of the application in the objectMap:

addObject(identifier)	:	adds a new object to the application
getObject(identifier)	:	returns the reference to the object
containsObject(identifier)	:	returns true or false
removeObject(identifier)	:	removes the identifier from the application
numberOfObjects()	:	returns the number of defined objects
clearObjects()	:	removes all objects of the application
getAutoId()	:	returns internally generated identifier

Java code:

```
import java.util.* ;

public class App{
  static HashMap objectMap  = new HashMap()                    ;
  ....

  //....Handle the objects of the application.....................

  public static AppObject addObject(AppObject object){
    if (containsObject(object.id)){
  Z.putS(" *** Application already contains Object ") ;
  Z.putS(object.id + " ***")                          ;
  return object                                       ;}
    return (AppObject)objectMap.put(object.id,object)     ; }

  public static AppObject getObject(String objectId)
  { return (AppObject)objectMap.get(objectId); }

  public static  boolean containsObject(String objectId)
  { return objectMap.containsKey(objectId); }

  public static AppObject removeObject(String objectId)
  { return (AppObject)objectMap.remove(objectId); }
  ....
}
```

4.3.4 Class AppObject

AppObject is the super class of all objects of the finite element application. The constructor AppObject(identifier) enters the object in the object map of class App(lication). The class implements the interface Comparable, so that objects of the application can be ordered lexicographically.

Java code:

```
package service ;

public class AppObject implements Comparable{
  public String id      ;

  public AppObject(String id){
    this.id = id        ;
    App.addObject(this) ;}

  public String getId() { return id; }

  public int compareTo(Object object){
    String s = ((AppObject)object).id ;
    return id.compareTo(s)            ;
  }}
```

4.4 Implementation of the model components

The component objects comprising the finite element model have to describe the structural system, the influences on the system, and the behaviour of the system, in the way stipulated by the finite element theory. Component objects for the thin plate model developed in chapter 3 are classified below.

Interfaces: Each of the classes implements a Java interface with the name: `Ic` prefixed to the class name. A Java interface is an abstraction of equivalent class functionality. As such it provides the capability to deal with instances of classes implementing an interface at a more general level of equivalence. This generalisation is convenient, e.g. in finite element models which allow for a number of different structural components. Finite element theories for different structural elements yield similar components, but with differences in the specifics. A node for a solid element, for example, has three position coordinates (x_1, x_2, x_3) and three translational degrees of freedom, compared to the plate node's two position coordinates (x_1, x_2), and one translational and two rotational degrees of freedom. The two node types are equivalent in the sense that both function as nodes in a finite element network, but differ in the specifics of their attributes. Equivalent behaviour of the two node types is then laid down at the interface level.

`Ic`-interfaces of the component objects: The `Ic`-interfaces implemented by the classes described below provide all the methods required to deal with the component objects. Consequently, access to component objects is realised at the level of their respective interfaces. The methods specified in these interfaces may be grouped as follows:

- `set` methods : By convention, these methods allow the specification/modification of object variables (attributes) at any time after the object has been instantiated. One noteable exception is the method setReferences(), which establishes the connections with associated objects by mapping the external identifiers of these objects to their respective references, as explained in section 4.3.2.
- `get` methods : By convention, these methods allow the retrieval of object variables.
- `compute` methods : Through these methods objects can be instructed to compute certain entities which are needed in the execution of the finite element algorithms.
- `print` methods: These methods provide alpha-numeric output of information encapsulated in object variables.

In the descriptions of the component classes below, only key methods from the interfaces are highlighted.

Class Node: Each object of this class describes a node of the finite element net. Nodes are constructed as follows:

- Node(identifier, firstIndex, x1, x2)
- Node(identifier, firstIndex, x1, x2, localSystem)

Nodes are handled through methods specified in interface IcNode. Key methods from the interface are:

- void setSystemIndices(firstIndex) : Sets the indices of the nodal variables in the system equations to firstIndex, firstIndex+1,...This method may be used to optimise the profile structure of the system equations.
- int[] getSystemIndices() : Returns the indices of the nodal variables in the system equations.
- void setReferences() : Establish reference-connections to objects via their identifiers.

Class Local: Each object of this class describes a local coordinate system in the finite element net. Local systems are constructed as follows:

- Local(identifier, theta)
- Local(identifier, sinTheta, cosTheta)

Local systems are handled through methods specified in interface IcLocal. Key methods from the interface are:

- void setTrigonometry(double angle) : Sets the angle (in degrees) of rotation from the global axis x1 to the local axis y1 and computes the rotation matrix R.
- void setAngle(double sin, double cos) : Sets the rotation matrix R and computes the angle of rotation.

Class Support: Each object of this class describes one support of the plate, which is associated with a specific node of the finite element net. Supports are constructed as follows:

- Support(identifier, nodeIdentifier, supportCode)
- Support(identifier, nodeIdentifier, status, status, status, value, value, value)

Supports are handled through methods specified in interface IcSupport. Key methods from the interface are:

- void setReferences() : Replaces all object identifiers by object references.

Class Material: Each object of this class describes a material of the plate. This material can be specified for several elements of the finite element net. Materials are constructed as follows:

- Material(identifier, modulusOfElasticity, poissonRatio)

Materials are handled through methods specified in interface IcMaterial. Key methods from the interface are:

- double[][] computeElasticityMatrix() : Computes the matrix which relates the states of stress and of strain : stress = matrix * strain.

Class Element: Each object of this class describes a triangular, thin plate bending element of the finite element net. Elements are constructed as follows:

- Element(identifier, nodeId1, nodeId2, nodeId3, materialId, thickness)

Elements are handled through methods specified in interface IcElement. Key methods from the interface are:

- void setReferences() : Establish reference-connections to objects via their identifiers.
- double[][] computeStiffnessMatrix() : Computes the element stiffness matrix. Numerical integration using the Gauss method is used. The method loops the integration points of the element, computes the strain interpolation matrix $\mathbf{B}$, as well as the product $\mathbf{B}^T\mathbf{EB}$, and adds the scaled product to the stiffness matrix.
- double computeDisplacementAtPoint(double z1, double z2, double z3) : Computes the state of displacement (u3,a1,a2) at the point in the element with the local coordinates z[].
- double[][] computeMomentAtPoint(double z1, double z2, double z3) : Computes the state of moment (m1,m2) at the point in the element with the local coordinates z[]. The element displacement vector is multiplied with the strain interpolation matrix $\mathbf{B}$, and the resulting strain state vector with the elasticity matrix $\mathbf{E}$ of the material. The stress state is scaled to yield the coordinates of the moment state.
- double[] computeShearAtPoint(double z1, double z2, double z3) : Computes the shear stress resultants q31,q32 at the point in the element with the local coordinates z[].

Class NodeLoad: Each object of this class describes a concentrated load at a node of the finite element net. NodeLoads are constructed as follows:

- NodeLoad(identifier, nodeIdentifier, qc3, mc1, mc2)

NodeLoads are handled through methods specified in interface IcNodeLoad. Key methods from the interface are:

- void setReferences() : Establish reference-connections to objects via their identifiers.

Class LineLoad: Each object of this class describes a line load between two adjacent nodes of the finite element net. LineLoads are constructed as follows:

- LineLoad(identifier, startId, endId, q3)
- LineLoad(identifier, startId, endId, q31,q32)
- LineLoad(identifier, startId, endId, q3, m1,m2)
- LineLoad(identifier, startId, endId, q31,m11,m21,q32,m12,m22)

LineLoads are handled through methods specified in interface IcLineLoad. Key methods from the interface are:

- void setReferences() : Establish reference-connections to objects via their identifiers.
- double[] computeLoadVector() : Computes the concentrated node loads due to the line load and returns the reference to the load vector. The load vector is computed in three stages. First, a local coordinate system is associated with the line load. Then the load vector is computed in the local coordinate system. Finally, the load vector is rotated to the global coordinate system and from there to the node coordinate systems of the system equations. The two methods below are used in these computations.

- IcLocal computeLocalSystem() : Computes the local coordinate system of the line, constructs the coordinate system with an internal identifier, stores its identifier and reference in the line load object and returns the reference to the local coordinate system.
- public double[] rotateLoadVector() : Rotates the concentrated node forces from the local coordinate system of the line element to the global coordinate system.

Class AreaLoad: Each object of this class describes a transverse distributed load which is applied to one of the elements of the finite element net. AreaLoads are constructed as follows:

- AreaLoad(identifier, elementIdentifier, p)
- AreaLoad(identifier, elementId, p1, p2, p3)

AreaLoads are handled through methods specified in interface IcAreaLoad. Key methods from the interface are:

- void setReferences() : Establish reference-connections to objects via their identifiers.
- double[] computeLoadVector() : The load vector for an area load is computed in the global coordinate system. The method loops the integration points of the element, computes the displacement interpolation vector **s**, and adds the scaled vector to the load vector. The vector is then rotated into node coordinate systems of the system equations with method rotateLoadVector().
- double[] rotateLoadVector() : Rotates the load vector from the element coordinate vector to coordinate systems used in the system displacement vector.

4.5 Implementation of the model

The classes of component objects of the finite element model were described in section 4.4. A set of objects can be instantiated which describe the loadbearing system and boundary conditions of the structure. In order to study the structural response, the finite element algorithms described in chapter 3 have to be executed. This is achieved through the addition of classes Model, Analysis and Equation. An object of class Model is used to manage the component objects of the finite element model. An object of class Analysis directs the procedures required to compute the behaviour variables of the Model, in the process using an object of class Equation.

4.5.1 Class Model

Scope: Each object of the class describes one finite element model for plate bending. The model primarily consists of HashSets with the references of the model components, and of methods to handle the HashSets. In addition, the alpha-numeric output of the model components and their behaviour is controlled with the print methods of the model.

Components: The model contains a HashSet for objects of each of the classes described in section 4.4. The empty sets are defined in the constructor of the model.

- HashSet nodeSet : Set of references of Node objects.

- HashSet localSet : Set of references of Local (coordinate system) objects.
- HashSet supportSet : Set of references of Support objects.
- HashSet materialSet : Set of references of Material objects.
- HashSet elementSet : Set of references of Element objects.
- HashSet nodeLoadSet : Set of references of NodeLoad objects.
- HashSet lineLoadSet : Set of references of LineLoad objects.
- HashSet areaLoadSet : Set of references of AreaLoad objects.

Handling: The method add(object) is used to add an arbitrary component to the model. The argument datatype is used to detect the appropriate HashSet to which the object is then added. For the removal of components from the model, separate methods are provided for each HashSet, for instance removeLocal(identifier).

Output: Output of information, as explained in section 4.2, is done through the model's print() methods. Output is done on a per set basis for sets of component objects. Before a set's information is printed, the elements are arranged in a sequence. A sequential storage structure[1], the ArrayList of the package java.util is used. This sequence is then lexicographically sorted according to the identifier property of the component objects. For the purpose of identifier sorting, the superclass AppObject of all component classes implements the interface Comparable. Since all component objects have an identifier the ordering relation is total[2]. Furthermore, the lexicographical ordering relation is strict[3]. Consequently the sorted sequence is unique[4]. The sequence is traversed and the appropriate print method, specified in the `Ic`-interfaces implemented by the component clases, is called for each element in the sequence.

4.5.2 Class Analysis

Scope: Each object of this class is used to analyse a particular model. An object of class Equation, described in section 4.5.3 below, is used to store the system matrix and system vector, and solve the system equations. The central method perform() controls the steps of the analysis:

```
void perform() throws FemException, AlgebraicException{
  setReferences()          ;
  computeSystemMatrix()    ;
  computeSystemVector()    ;
  setStatusVector()        ;
  system.decompose()       ;
  system.solve()           ;
  saveResult()             ;
  isPerformed = true       ;
}
```

[1] cf. section 4.1.4.1
[2] cf. section 4.1.2
[3] cf. section 4.1.2
[4] cf. section 4.1.4.2

setReferences(): This method traverses all HashSets of the model. For every object in the sets, the method setReferences() is called. This method determines the references of all objects whose identifiers are contained in the object. If the object corresponding to an identifier does not exist, an exception is thrown. After the execution of setReferences(), all objects of the model are connected with references.

computeSystemMatrix(): This method computes the coefficient matrix of the system equations. It first determines the dimension of the system equations by finding the largest system index at the nodes. It then traverses the element set and constructs the profile of the system equations. In a second traverse of the element set, it computes the element matrices and adds them to the system equations.

computeSystemVector(): This method computes the system load vector. The contributions of the node loads, line loads and area loads are determined in separate traverses of their HashSets. The node loads are added directly to the system equations, since they are referred to the node coordinate system. The line load vector, and the area load vector are rotated to the node coordinate systems of the system equations before they are added to the system equations.

setStatusVector(): This method sets the status vector[1] of the system equations. In a traverse of the supports of the model, the status is set to true for all prescribed displacement components.

decompose(): This is a method of class Equation, which is described in section 4.5.3 below. The method decomposes the system coefficient matrix by the method of Cholesky.

solve(): This is also a method of class Equation. It solves the system equations for the prescribed displacements and the current system load vector. The results are the displacements in the vector primal[], and the reactions in the vector dual[].

saveResult(): This method stores the displacements from the system displacement vector to the nodal displacement vectors. It also stores the reactions from the system reaction vector to the supports. Once the element nodal displacements are available, strains, stresses and stress resultants inside the element can be computed as described in section 3.2.4.

[1] cf. section 4.5.3

4.5.3 Class Equation

The computation of the system matrix and system load vector were described above. The finite element system equations 3.2.45 have to be solved for the unknown coordinates of the system displacement vector and reaction vector. In practical finite element analyses, the number of unknowns can be high. It is of the utmost importance that the storage and solution of the system equations are optimised. The special datastructure and solution procedure used in class Equation were developed to use resources as economically as possible.

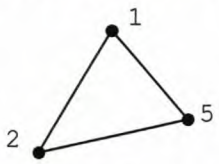
4.5.3.1 Profile of the system matrix

In class Analysis, the computation of the system matrix is implemented by computation and addition of element contributions in a loop over the elements:

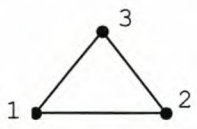
$$[K]_s = \sum_e [T]_e^T [R]_e^T [k]_e [R]_e [T]_e = \sum_e [T]_e^T [K]_e [T]_e \qquad (3.2.30')$$

In the computation of an element's contribution $[T]_e^T [K]_e [T]_e$, the multiplication with the element topology matrix $[T]_e$ is substituted by a mapping of element indices to system indices. The system matrix is symmetrical and has a profile structure. These properties are exploited in the solution of the system of equations. For this purpose the system matrix is stored by row, from the profile element to the diagonal element. The profile is determined automatically.

Mapping of the indices: The example below shows the global indices 2,5,1 and the local indices 1,2,3 of the state variable vectors at the nodes of a triangle:



global



local

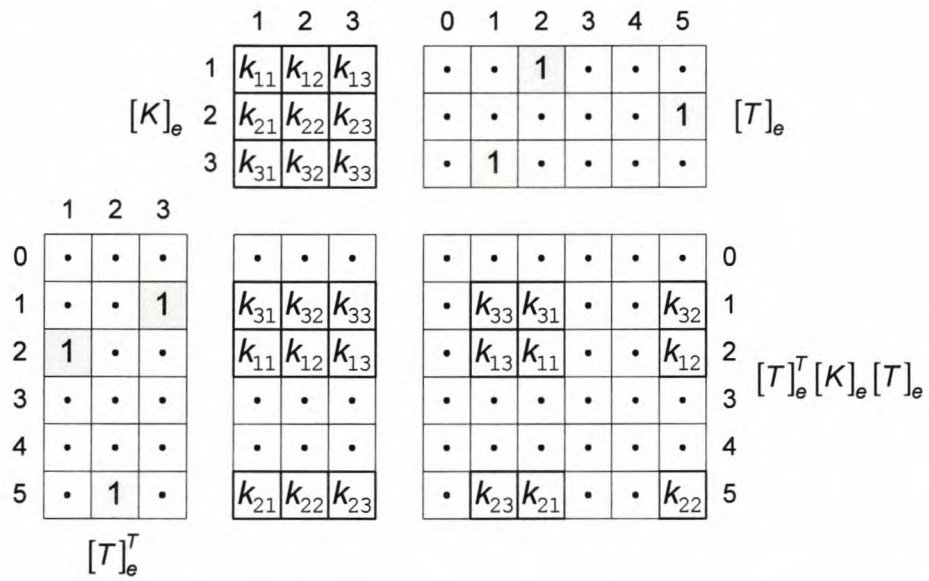
At each node, the state variable vector is:

$$\{u\}_{n(i)} = \begin{Bmatrix} u_3 \\ \alpha_1 \\ \alpha_2 \end{Bmatrix}$$

The element has following topology matrix:

$$\{u\}_e = \begin{Bmatrix} \{u\}_{n(1)} \\ \{u\}_{n(2)} \\ \{u\}_{n(3)} \end{Bmatrix} = \begin{bmatrix} \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & 1 & \cdot & \cdot & \cdot & \cdot \end{bmatrix} \begin{Bmatrix} \{u\}_{n(0)} \\ \{u\}_{n(1)} \\ \{u\}_{n(2)} \\ \{u\}_{n(3)} \\ \{u\}_{n(4)} \\ \{u\}_{n(5)} \end{Bmatrix} = [T]_e \{U\}_s$$

The coefficients of the element matrix $[K]_e$ are k_{im}. Then, explicit multiplication of $[T]_e^T [K]_e [T]_e$ leads to the following result:



The multiplication $[T]_e^T [K]_e [T]_e$ of the element matrix $[K]_e$ with the topology matrix $[T]_e$ distributes the coefficients of $[K]_e$ in the system matrix $[K]_s$. This distribution can be implemented with a map $m : \{1,2,3,\} \rightarrow \{m_1, m_2, m_3\}$ with $m(i) = m_i$. The map m is described by an index vector. The coefficient k_{is} of $[K]_e$ is added to the coefficient $k_{m_i m_s}$ of $[K]_s$. The example above has the index vector:

i	1	2	3
m_i	2	5	1

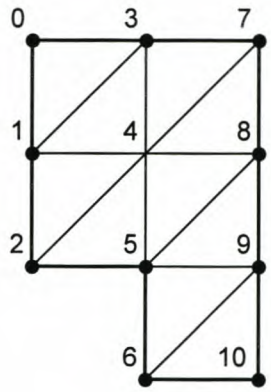
$$k_{23} \in [K]_e \rightarrow k_{m_2 m_3} = k_{51} \in [K]_s$$

Profile vector of the system matrix: The natural stiffness matrix of the finite element system has a profile structure if the system indices of the nodal degrees of freedom are assigned in a proper sequence. The profile structure can be exploited to minimise storage requirements and reduce the computational effort required for Cholesky decomposition of the coefficient matrix. Since the system matrix is symmetric, the profile also has a symmetric shape, which can be described by a single profile vector. For the i^{th} equation, profile[i] contains the index of the first non-zero element of the coefficient matrix. As an example, the system matrix and profile vector of the finite element net below is shown:

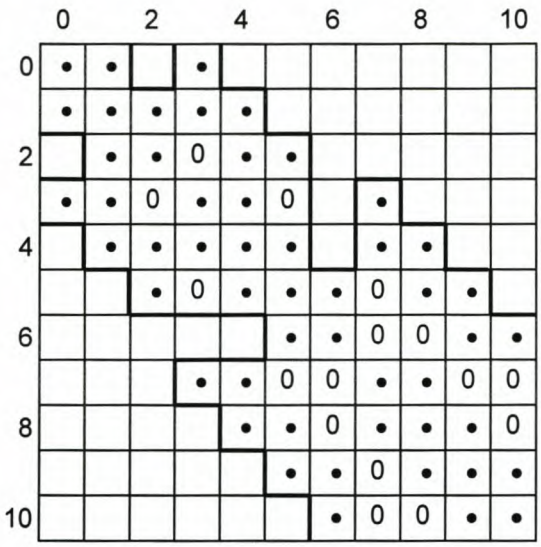
Profile vector: The diagonal term of the system matrix $[K]_s$ is always non-zero. Consequently the i^{th} row of the profile vector is set to the initial value profile[i] = i. The elements of the net is then investigated in a loop. For a general element e the index vector is determined. In a double loop over the index vector, the system indices m_i and m_s is read for

each element $k_{is} \in [K]_e$. If the profile of $[K]_s$ in row m_i is greater than m_s, then profile[i] = m_s is set.

$$\bigwedge_i \bigwedge_s (\text{profile}[m_i] > m_s \Rightarrow \text{profile}[m_i] = m_s)$$



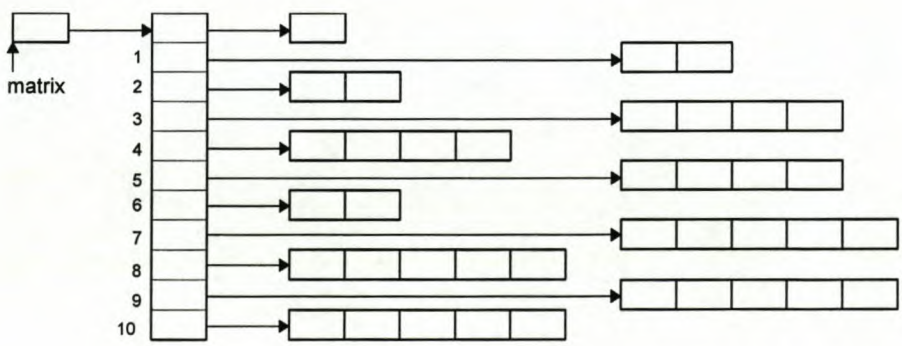
Net with system indices



System matrix

profile[]

Datastructure: The system matrix $[K]_s$ is stored by row as an Array. The vector for row i contains, in sequence, the coefficients from column profile[i] to column i. The vector for row i contains $(i - \text{profile}[i] + 1)$ coefficients. Since the coefficients of value zero at the start of the row are not stored, the coefficients of $[K]_s$ are not accessed by normal matrix indexing matrix[i][h]. For this purpose class Equation has the methods getValue(row,column), putValue(row,column,value) and addValue(row,column,value).The reference `matrix`, and the reference vector `matrix[]` for the rows of $[K]_s$ are assigned in the constructor of class Equation. The row vectors are allocated by the method allocateMatrix() of class Equation after the profile vector has been created. Storage of the system matrix of the example above is shown:



4.5.3.2 Solution of the system equations

Gauss elimination: The standard technique for the solution of a linear system of equations is Gauss elimination. This technique can be applied to general linear systems of equations. If the equations have a unique solution, it effectively reduces the coefficient matrix to upper triangular form, after which the unknown coordinates are computed in a backward sweep. Reduction of the left (coefficient matrix) and right hand (load vector) sides may be performed separately, which is advantageous when solutions to different values of the right hand side are sought. The coefficient matrix of the finite element system equations 3.2.45 is symmetric and positive definite. These special characteristics are taken into account by the method of Cholesky.

Cholesky decomposition: The method of Cholesky can be applied to systems of equations which have a symmetric, positive-definite coefficient matrix. Such a matrix can be decomposed into the product of a lower triangular matrix and its transpose. Consequently, the system of equations with coefficient matrix $[A]$ can be written as:

$$[A]\{x\} = \{b\}$$
$$[L][L]^T\{x\} = \{b\}$$
$$[L]\{y\} = \{b\} \qquad \text{in which} \qquad [L]^T\{x\} = \{y\}$$

The coordinates of the vector $\{y\}$ can be obtained from the equation $[L]\{y\} = \{b\}$ by a forward sweep. The coordinates of the vector of unknowns, $\{x\}$, then follows from $[L]^T\{x\} = \{y\}$ by a backward sweep. In the process the right hand side ($\{b\}$) is not affected, and the storage requirement is limited to what is necessary for a lower triangular matrix. Since the coefficient matrix $[A]$ is symmetric itself, only its lower trianglar part has to be stored as well. During decomposition of the coefficient matrix, the values in $[A]$ can be overwritten.

System equations: The system equations are of the form: $[A]\{x\} = \{c\} + \{y\}$. The coefficients of $[A]$ and $\{c\}$ are known. If a coordinate in $\{x\}$ is unknown, the corresponding coordinate in $\{y\}$ is prescribed, and vice versa. The unknown coordinates of $\{x\}$ can be grouped at the top of the vector, and the prescribed values at the bottom:

$$\begin{bmatrix}[A]_{11} & [A]_{12}\\ [A]_{21} & [A]_{22}\end{bmatrix}\begin{Bmatrix}\{x\}_1\\ \{x\}_2\end{Bmatrix} = \begin{Bmatrix}\{c\}_1\\ \{c\}_2\end{Bmatrix} + \begin{Bmatrix}\{y\}_1\\ \{y\}_2\end{Bmatrix} \quad \text{in which} \quad \begin{matrix}\{x\}_1, \{y\}_2 & \text{unknown variables}\\ \{x\}_2, \{y\}_1 & \text{prescribed variables}\end{matrix} \qquad (4.5.1)$$

The first equation in (4.5.1) yields:

$$[A]_{11}\{x\}_1 = \{h\}_1 \qquad \text{with} \qquad \{h\}_1 = \{c\}_1 + \{y\}_1 - [A]_{12}\{x\}_2 \qquad (4.5.2)$$

Equation (4.5.2) can be solved, using Cholesky's method, for the unknowns $\{x\}_1$. Substituting the computed $\{x\}_1$ into the second equation of (4.5.1) yields the unknowns $\{y\}_2$:

$$\{y\}_2 = [A]_{21}\{x\}_1 + [A]_{22}\{x\}_2 - \{c\}_2 \tag{4.5.3}$$

Solution with status vector: Rearrangement of the system of equations to the form (4.5.1) is costly, and, more importantly, it destroys the profile structure of the coefficient matrix. The solution procedure is consequently executed with the aid of a status vector $\{s\}$. The status vector coordinate $\mathbf{s}(i)$ is `true` when $\mathbf{x}(i)$ is prescribed and $\mathbf{y}(i)$ unknown, and `false` when $\mathbf{x}(i)$ is unknown and $\mathbf{y}(i)$ prescribed. Implementation of the solution procedure follows the steps:

1. In the rows with status `false`, the substitution (4.5.2) is executed.
2. In the rows and columns with status `false`, the decomposition $[A]_{11} = [L][L]^T$ is performed
3. In the rows with status `false`, the unknowns $\{x\}_1$ are determined by the forward and backward sweep.
4. In the rows with status `true`, the unknown reactions $\{y\}_2$ are determined by substituting $\{x\}$.

4.6 Class diagram

Class model: In object-oriented models the class serves as an abstraction of equivalent objects, as explained in section 4.1.1. The class model of the application is a structured set of classes. The structure is laid down in a set of binary relations between the classes. Each relation between two classes represents relationships between two corresponding sets of equivalent objects of the model.

Class diagram: The class diagram is a graph which represents the class model. The classes are the vertices, and the relationship between them the edges of the graph. The classes comprising the thin plate finite element model, and the relationships between them, were described in sections 4.3 through 4.5. The class diagram of the application is shown in figure 4.1 below. Contiguous classes are assembled into packages as shown.

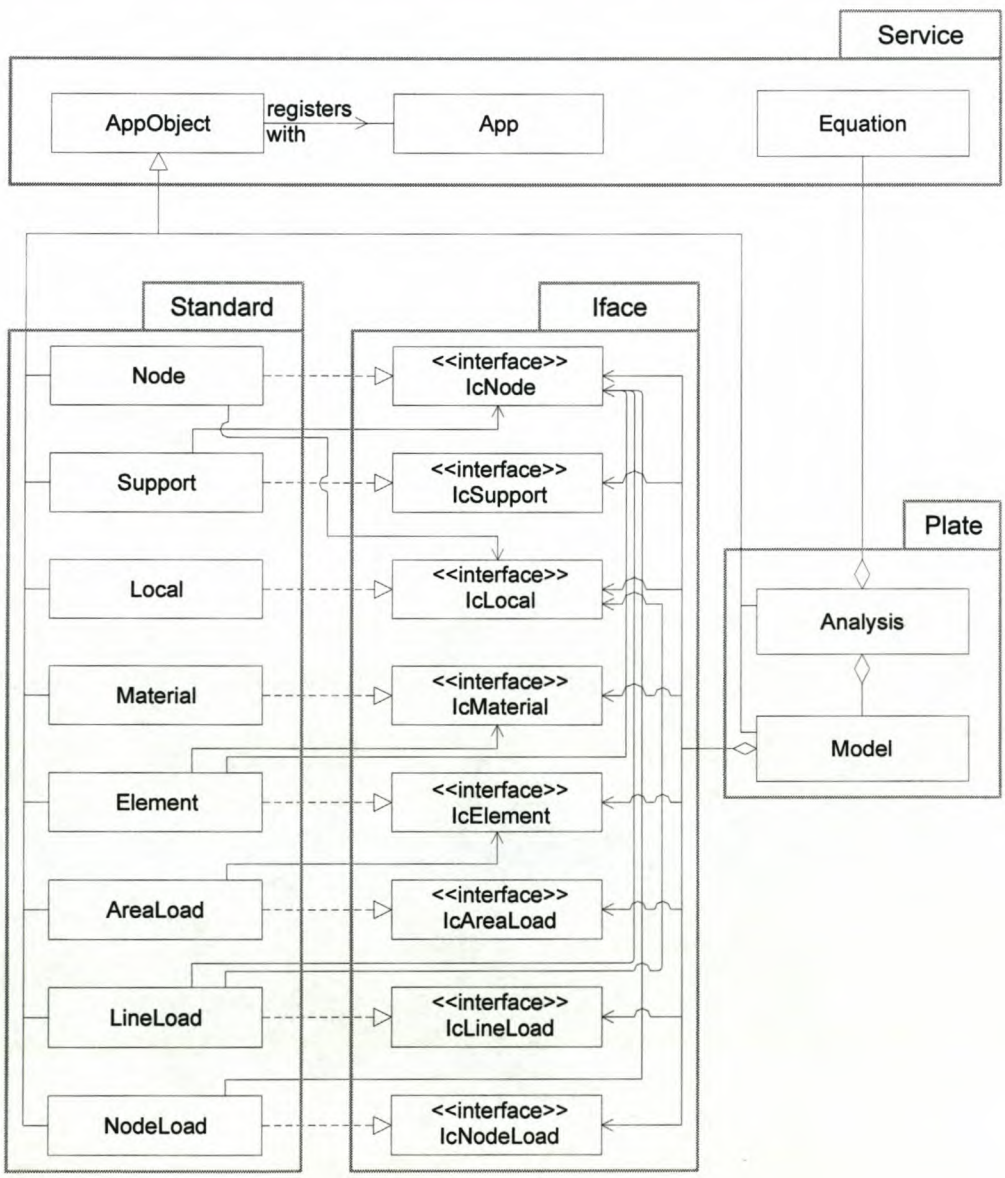


Figure 4.1: Application class diagram

5 Scope of models for distributed collaboratories

The algorithmic basis of numerical structural analysis, and the development of an object-oriented finite element model, designed for use in local environments, which has the necessary components and functionality to execute the finite element algorithms, were described in the previous two chapters.

Distributed design of built facilities implies that finite element analyses have to be performed in a distributed communication network, with a number of engineers collaborating in the creation, modification, and analysis of structural models. The specification of an object oriented model, called the distributed analysis model, which is capable of supporting a structural analysis collaboratory, is developed in this chapter.

Aspects of the implementation of the specifications are addressed in following chapters.

5.1 General scope of the distributed analysis model.

In this section the setting of a distributed structural analysis collaboratory is sketched. It is shown that the distributed analysis model comprises two areas of functionality, one which provides for the technical aspects of structural analysis, while the other supports collaboration. Requirements of both of these groups of functions need to be taken into account in the specification the distributed analysis model. Individual aspects of the model are specified in subsequent sections. Various types of requirements and influences may be taken into account in a specification such as the one developed here. The point of view taken below is based on

- the special characteristics and requirements of structural analysis, and
- the requirements and limitations of the computer and network infrastructure which supports the distributed collaboration.

Distributed structural analysis collaboratory: Consider a project to erect a built facility. The core task of structurally designing the facility requires a number of structural analyses to be undertaken. For this purpose a distributed structural analysis collaboratory is formed. The collaboratory comprises three components, namely the structural analysis tasks, the engineers who collaboratively perform these tasks, and the distributed analysis model they employ in the process.

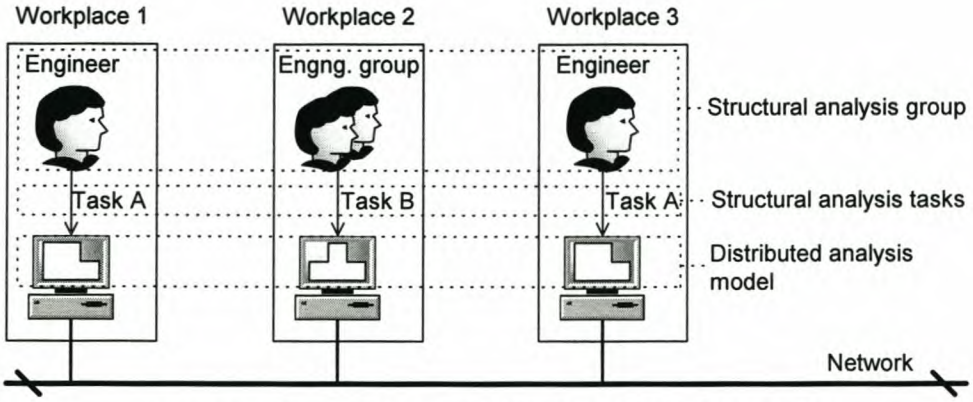


Figure 5-1: Distributed structural analysis collaboratory

1. **Structural analysis tasks**: It is assumed that the different analysis tasks are assigned, and that the engineers who execute these tasks know what their respective roles and responsibilities are. The tasks are to be executed according to the structural analysis schedule, which depends on the structural design and project schedules. The analysis schedule may require that certain tasks be performed parallel in time, while others may be performed sequentially.
2. **Structural analysis group**: A group of engineers, the structural analysis group, are collectively responsible for the execution of the tasks mentioned above. The composition of the group depends on the project, and it is usually formed ad-hoc. Members may come from different organizations. The group consists of subgroups and individuals whose workplaces are physically separated, but connected by a computer network (typically the Internet). The separation may be large, i.e. not limited to the same building, city or country, precluding normal modes of cooperation.
3. **Distributed analysis model**: A set of computer-based parameters and functions which enable the group members to efficiently perform their respective analysis tasks, at their respective locations, while collaborating as much as possible is called the distributed analysis model. Apart from analysis functionality, which is extended from that which was developed for the local environment, the distributed analysis model needs a group of functions which supports and administers the cooperation in the distributed collaboratory. Essentially, the distributed analysis model supports collaborative structural analysis by bringing together process and product in the distributed environment[1]:

- ***Process:*** Finite element structural analysis (of thin plates) was developed as an application in chapter 4. The essential analysis functionalities contained therein are extended and distributed to the members of the analysis group in order to support the analysis process while making use of cooperative resources.

[1] see section 1.2.1

- ***Product:*** The parameters of an analysis contain the essential information which describe the product (i.e. the structural model) being analysed. These parameters are distributed amongst the members of the analysis group in order to support the collaborative processing of shared parameters. This requires making project analysis information available to members of the analysis group, enabling them to selectively upload and assimilate such information.

Information, knowledge and action: In all of the core engineering tasks, information is processed. The information evolves to knowledge through its interpretation by trained and experienced engineers, and the knowledge forces the engineers to take appropriate actions. In this dissertation the focus is not on the support of knowledge and actions, but on the collaborative processing and exchange of information. The necessary distribution of parameters and functionality source a large number of issues, which are discussed in following sections.

5.2 Distribution of parameters

Common parameters and distribution: The processing and consistent management of common parameters are central aspects of any form of group work, be it at the same location or in a distributed environment. A large spectrum of parameters exist which have to be shared amongst groups involved in the design of a built facility. The question what exactly comprises the essential structural analysis parameters is crucial.

Collaboration parameters of the project: The distributed analysis model needs to be aware of the structural analysis group, the places from which they work, and the analysis information they share in order to support the collaborative processing and exchange of this information in a distributed communication network. The set of objects which describe the engineers, their workplaces, the locations of analysis information, etc. is called the collaboration parameters of the project. The management of the collaboration parameters is dictated by the architectural layout of the collaboratory, software implementation details, and decisions regarding the distribution of the analysis parameters of the project, and should be transparent to the members of the analysis group.

5.2.1 Analysis parameters of the project

Essential parameters: The focus of the distributed analysis model is only on the essential information requirements of distributed structural analysis. In chapter 4 the essential parameters for the analysis of thin plates in a local environment have been identified as instances (objects) of the classes `Node`, `Local`, `Support`, `Material`, `Element`, `NodeLoad`, `LineLoad`, `AreaLoad`, `Model`, `Analysis` and `Equation`. It was shown that using nothing more than an appropriate set of these objects, it is possible to analyse a thin plate of arbitrary shape and boundary conditions. The application developed in chapter 3 was kept as simple as possible by allowing for the analysis of one model only. The system

equations were built and solved by employing an object of class `Analysis`. In a local environment a user can easily manage models, e.g. by storing files related to a model in a particular directory. In the distributed collaboratory, models are used as logical units of information. For this purpose the classes `Model`, `Analysis` and `Equation` are modified: Class `Analysis` becomes a purely functional class, i.e. it does not carry information, but provides the necessary analysis functionalities. An object of class `Equation` is added as an attribute to class `Model` in order to complete the model as an information unit. As a result, objects of the following classes are processed and exchanged in the structural analysis collaboratory:

- `Model, Equation, Node, Local, Support, Material, Element, NodeLoad, LineLoad, AreaLoad.`

The set comprising these objects, for the project as a whole, is called the analysis parameters of the project.

Specification: The distributed analysis model manages the analysis parameters of the project as a logical unit. All members of the analysis group should be able to access the analysis parameters, subject to project specific controls, from their respective workplaces.

5.2.2 Work pattern

An analysis task typically involves three phases, as explained in chapter 4. The way in which analysis parameters are used in each of these phases leads to important requirements for their distribution.

Modeling phase: This phase entails the definition and modification of the geometry, material, loading and support conditions of the finite element model. The analysis parameters involved are the `Node, Local, Element, Material, NodeLoad, LineLoad, AreaLoad` and `Support` components of the `Model` under construction. The phase is characterised by a large number of small changes to the stated parameters, and is effected by the engineer in sessions which involve graphical interaction to a great extent. In a collaboratory it is to be expected that some of the modifications to the model involve the assimilation of existing analysis parameters made available by a cooperative source. Structural analysis practise requires two modes in which an existing parameter or set of parameters are used:

- *Consistent mode*: A consistent parameter has the same state in all models in which it is consistent. If a user has the necessary rights to employ a parameter in consistent mode, and needs to change its state, the changes have to be reflected in other models in which the parameter is used.

  Example: Two members of the analysis team collaborate in the analysis of a structure. One is investigating the ultimate limit state, the other the serviceability limit state. (Different loading conditions apply to the two states). Since the two models should use exactly the same geometry, material, and kinematic boundary conditions, the shared parameters which describe these entities, i.e. `Node, Local, Material, Element` and `Sup-`

`port` parameters, should be used in consistent mode to ensure that fundamental changes to any of these parameters are reflected in both models.

- *Free mode:* If a user employs a parameter in free mode, changes to the parameter only affect the model under construction. This mode is useful in the collaborative environment since two parameters may be related without having the same state. *Example:* Consider the example listed above. If some team member has to study the dynamic properties of the structure, he might choose to model distributed mass effects by adapting the floor slab density in certain areas. New versions of the affected `Material` parameters are required to model the changed thickness. However, these changes should not be reflected in the other models. Therefore these parameters should be used in free mode.

Modeling may take anything from hours to days, depending on the complexity of the model.

Algorithmic phase: During this phase the algorithms which assemble and solve the system equations are executed. The `Equation` parameter of the model is constructed in the process. The phase is computationally intensive, and interaction is either limited or absent. Its duration depends on the computer's processing speed and the number of operations required to reach the solved state.

Interpretation phase: The various results, e.g. displacements, stresses, and stress resultants at different locations of the model are studied and interpreted. The analysis parameters involved are the `Node`, `Local`, `Element`, and `Support` components of a `Model`, as well as the primal and dual result vectors of its `Equation`. As a rule the results are displayed graphically, but alpha-numeric format is also used. Initial interpretation is usually done in interactive worksessions. In an analysis collaboratory, engineers at different locations need to view and discuss the same results. It is important to note that interpretation itself does not bring about fundamental changes to the analysis parameters, so all parameters are used in free mode. Interpretation of complex models may be time consuming.

Specification: The analysis parameters have to be distributed in the mode chosen by the user, i.e. either consistent mode or free mode, subject to relevant access controls. Furthermore, the interactive nature of the modeling and interpretation phases, and the computationally intensive nature of the algorithmic phase have to be considered.

5.2.3 Information units and granularity

Model: The work pattern described above stresses the importance of models in the analysis of the structure under investigation. The finite element `Model` is an innate information unit of structural analsysis. The project analysis parameters have to be managed in such a way that a model can be managed as a unit.

Component sets: In analysis practice it is typical that two models differ only slightly, e.g. in their respective support or loading conditions, the details of the material, etc. As a result it is

essential to store and exchange component sets belonging to a model, e.g. the set of `Support`, `AreaLoad` or `Material` parameters.

Individual parameters: In order to effectively support the construction of a new model using parameters of existing models, access and selection of analysis parameters should be supported at the level of individual parameters (e.g. `Material` parameters).

Specification: Analysis models and their component sets should be identifiable and distributable to existing or new models at any workplace. Furthermore, distribution of parameters at the granularity of the individual parameter should be supported.

5.2.4 Access control

Each project analysis parameter is created by its owner who employs it in one or more models. Since single parameters may be exchanged in the collaboratory, an owner should be able to control access to each of the parameters he created, while taking account of the obligations he has as a member of the analysis group. Depending on the structure of the group, default access controls may be applicable. Different access controls may apply for the use of a parameter in consistent mode and free mode.

Specification: The owner of a parameter should be able to control access to the parameter. A suitable mechanism for default access controls should be provided. Setting different access controls for use of parameters in consistent mode and free mode should be possible.

5.2.5 Locking

The modeling phase described in the analysis work pattern above may require synchronous operations on the same set of analysis parameters by two or more members of the analysis group. The typical modeling phase is not short, and cannot be subdivided into short-duration transactions. Furthermore an analysis model may comprise a large number of parameters. As a result, the locking of parameters by a user for the duration of a modeling session cannot be tolerated, i.e. the traditional concept of a transaction cannot be applied in collaborative structural analysis. Long duration checkout of parameters by a user, without locking access to the parameters by other users is essential.

Specification: Existing project analysis parameters, to which a member of the analysis group has access rights, may not be locked by the actions of another member of the group. No restriction may be applied to the time that a user is allowed access to project analysis parameters.

5.2.6 Versioning

Parameter evolution: Long duration checkout of a parameter, without locking access to it by other users, changes the development of the parameter from being transaction dependent to being time dependent, i.e. it demands that the parameter be versioned.

Version interpretation: Since the mode in which a parameter is checked out may be either consistent mode or free mode, a twofold interpretation of different versions of a parameter is possible:

- *Consistent versions:* The properties of an existing parameter, checked out in consistent mode, may be considered and found to be incorrect. After the necessary corrections, a new version of the parameter supercedes the old version, i.e. the old version is invalid and should not be used further. Furthermore, all parameters that depend on the old version, e.g. models in which it was employed, have become invalid and should be updated.
- *Free versions:* A parameter may be checked out in free mode, to be employed in a model because it has a number of useful attributes. Some fundamental changes, however, may need to be effected, with the result that a new version of the object is created. This new version of the parameter, however, has no relation with the old version. It is a valid parameter in the current model, while the old version is still valid within the contexts in which it was used before.

Particular solution: Proliferation of versions, and difficulties to maintain the consistency of the project parameter base are known problems associated with long duration checkout. General solutions to these problems do not exist. However, by involving the user in the control of versioning, and in the maintenance of consistency, solutions which take the special characteristics of structural analysis into account may be developed.

Specification: The members of the analysis group have to control the versioning of the project analysis parameters. Furthermore, the consistency of the parameter base should be maintained through the active involvement of the users.

5.2.7 Persistent identification

The mechanism of using persistent external identifiers to manage the objects of an application was described in chapter 4. This mechanism will also play a key role in the distributed analysis model, which implies that a unique identifier is assigned to each of the project analysis parameters. Since parameters are also versioned, their identification for the purposes of distribution can be managed through a combination of their respective persistent identifiers and versions.

Specification: The identification of parameters in the distribution process should be based on a suitable combination of the persistent identifier and version of each parameter.

5.2.8 Worksession modes

Computer response time: From a computer user's point of view fast response times are always valued. In terms of computer support, the modeling and interpretation phases require a high degree of interactive response, and the algorithmic phase requires computing power. However, the acceptability of the actual response time is determined by the worksession mode.

Worksession mode: For the structural analysis collaboratory three worksession modes are applicable:

- Startup-Shutdown mode: At the start and end of a worksession it is normal for a startup-delay due to initialisation of the session, or a shutdown-delay due to clearup actions to occur. In this mode a waiting time of the order of a minute is acceptable.
- Interactive mode: The engineer is actively involved in modeling or interpretation of results and needs fast response. If the response times of interactive operations depended only on local resources, i.e. the workstation, the required response can be obtained through the installation of a suitably powered workstation. Response times in the collaborative environment, however, are critically dependend on the latency of network communication. The [WELD 1999] project measured the latencies (including packaging, transmission and unpackaging) listed in table 5-1 for the wide area network (WAN) used in the project. These results indicate that acceptable response times to interactive operations can only be realised in current wide area networks if the size of the information package which has to be transferred during the operation is small (in the case listed smaller than 100 kilobytes).

Package size (KB)	Latency (sec)
1	0,1
10	0,6
100	3
1 000	50
10 000	>100

Table 5-1: WAN latency

- Batch mode: If a certain set of parameters need to be transmitted very infrequently, e.g. once a week, and the need for the transmission can be predicted (i.e. planned for), the task can be executed in batch mode without loss of productivity. This is typical of file-based information exchange. It may serve a purpose on special occasions, but does not allow members of the analysis group to efficiently collaborate in the distributed environment.

Specification: The distribution of analysis parameters may be associated with a startup and shutdown delay. In interactive mode, however, the user has to be able to request uploading of parameters in package sizes of which the associated latency is acceptable to the user.

5.2.9 Parameter data volume

Volume prediction: The number of parameters required to analyse the structural behaviour of a built facility cannot be predicted systematically. An indication of parameter volumes is obtained by considering a square plate modeled as described in chapter 4. The plate is meshed with an even number of subdivisions in both directions. For three different subdivisions per side, the volume, in megabytes, of storage required by the different parameter sets in the model was determined. The relations between the number of degrees of freedom (N) of the model, and the storage volumes are shown in Figure 5-2 for the `Node` and `Element` parameter sets, and the model's `Equation`. The parameters `NodeLoad`, `LineLoad`, `Local` and `Support` have storage requirements comparable to `Node`, while `AreaLoad` parameters are comparable to `Element`. Storage requirements of `Node` and `Element` parameters increase linearly, and that of the `Equation` quadratically.

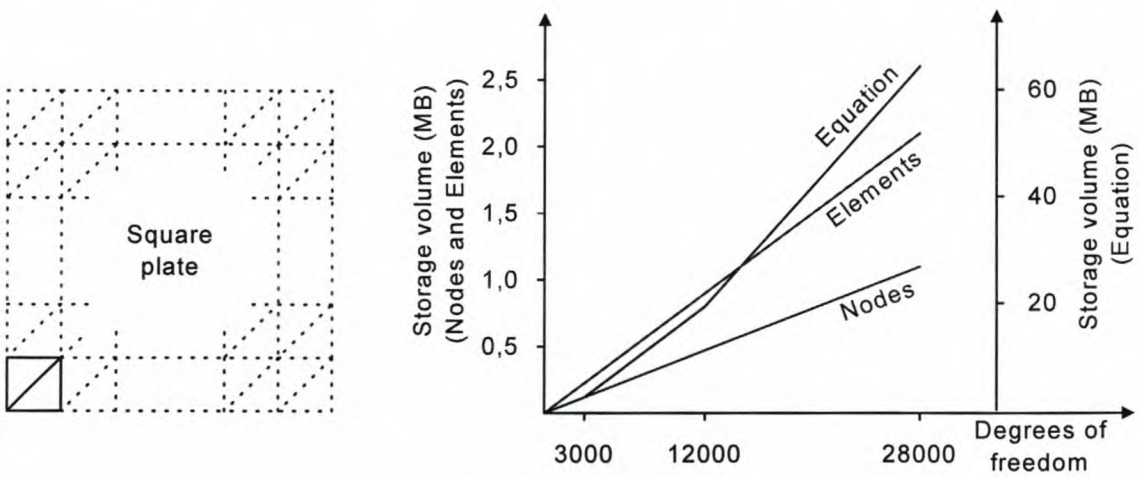


Figure 5-2: Storage volume of analysis parameters

Project storage requirements: The analysis of a typical building floor, with irregular layout and support conditions, cutouts, special loading patterns, etc. would require a model with number of degrees of freedom in the range between 3000 and 50000. This indicates that the storage requirements of the analysis parameters of a project might be several gigabytes for larger projects, which places a significant demand on the scalability of the project database and emphasises the question of its location.

Equations: A striking feature of figure 5-2 is the dominance of the model's `Equation` storage requirements. An `Equation` contains three attributes of value, i.e. the Cholesky factor of the stiffness matrix, the primal solution vector, and the dual solution vector. The storage volumes of the solution vectors are small compared to that of the Cholesky factor. For the three cases listed in figure 5-2, the storage requirement for each of the vectors is respectively 26KB, 101KB and 226KB versus the 2,6MB, 19,8MB and 65,8MB required by the Cholesky factors. Distribution involving transport over the network is quite plausible for the the solution vectors, but not for the Cholesky factor of the stiffness matrix.

Specification: The storage of the project analysis parameters should be undertaken in a database which allows effective management of significant volumes of data. Furthermore, the primal and dual solution vectors of the `Equation` should be available for distribution separate from the Cholesky factor.

5.2.10 Persistent storage

Information exchange: The distributed analysis model supports communication and cooperation amongst the members of the analysis group through the exchange of project analysis parameters. In order to support asynchronous, sequential collaboration, a group-wide information space has to be created around a database through which parameters can be exchanged via persistent storage and retrieval. The common database is called the Project Analysis Parameter Database (PAPD). Although synchronous, parallel collaboration does not require persistent storage of information during a work session, storage between work-sessions is necessary in any event. The role played by the PAPD is critical to the collaboration effort, since official exchange of parameters amongst members of the analysis group will only take place through the PAPD.

Database technology requirements: The PAPD is directly or indirectly involved in all the specifications developed in this chapter. The management of information, and the communication requirements in a distributed structural analysis collaboratory, however, demand additional considerations which pertain to the database technology employed in the PAPD:

1. Criteria regarding the general management of information in the collaboratory are stated in section 1.2.3. Satisfaction of those criteria is the task of the distributed analysis model, which in turn makes use of functionalities provided by the database. Three of the stated criteria are tightly linked to database technology:
   - ***Flexibility:*** Structural analysis is a creative task, and the distributed analysis model should not stifle collaboration by restricting members' resourceful management of parameters. Manipulation of parameter sets, and the creation of relations are key aspects enhancing productivity during the modeling and interpretation phases of an analysis. The forces and requirements which govern these actions, however, strongly depend on specific circumstances, and the particulars of the task under consideration, and cannot be effectively pre-programmed. Collaboration will be encouraged if the interface to the parameter base is flexible.
   - ***Media-suitability:*** The exploitation of mobile communication technology is, for the foreseeable future, subject to limited data transfer rates. This implies that effective collaboration using mobile technology is only possible if nothing more than essential information needs to be transmitted. Detailed selection of parameters, on the basis of flexibly specified engineering semantics, will make this important segment of the collaboratory viable.

- ***Navigability:*** The number of parameters in an analysis collaboratory is potentially large, and the task of finding relevant parameters should be supported. Strict navigation conventions and policies cannot account for personal preferences and specific project requirements, and have to be avoided. Customiseable support of searching and sorting parameters will significantly enhance user productivity and limit the occurrence of unsuccessful parameter transfer.

2. The volume of information generated in numerical structural analysis can be large As a result the PAPD has to scale well, i.e. it has to remain robust and efficient in spite of large data volumes.
3. The project's analysis information has to be kept for reference, and further processing, after the conclusion of the design and construction phases of the project. A widely supported, well-documented and preferably standardized interface to the PAPD would better serve towards the long-term accessibility and useability of the information.

Storage location: The storage and retrieval of parameters over a communication network are fundamental actions in the distributed collaboratory, and the productivity of the analysis group crucially depends on these actions. The response times, and the reliability of the service associated with the storage and retrieval of parameters in turn depends on the physical location(s) of the parameter base. The database structure could be monolithic or distributed, with database objects respectively residing in the filesystem of a single operating system, or in filesystems of network-connected multiple operating systems. From both the response time and reliability of service points of view, storage location is critical. Reliability of service is addressed in one of the criteria regarding the management of information in the collaboratory stated in section 1.2.3, namely that of standalone capability. It is considered to be very important to allow local work, at a specific workplace, to continue even if the communication link to the analysis group has been lost temporarily. This implies that a member of the group should be able to store the parameters of the active task in a local filesystem.

Specification: The project analysis parameter database should performantly support multi-user access and large data volumes. In distilled form, the database technology requirements described above express the need for a flexible, stable, and preferably standardised interface to the database. Standalone capability at a workplace, through local storage of parameters used in an active task, should be possible.

5.3 Distribution of methods

The distributed analysis model is the set of project analysis parameters and methods needed for processing these parameters in the distributed environment. Methods which provide standard analysis functionality in a local environment have to be extended to support the distributed analysis process. This process is a set of activities and their mutual dependencies, which collectively enable members of the analysis group to efficiently perform their respective analysis tasks while collaborating as much as possible. The methods of the distributed analysis model provide the functionality required to perform these activities.

5.3.1 Activation of references

The cornerstone of an analysis collaboratory is the ability to exchange parameters amongst the members of the analysis group. The capacity to introduce a parameter into a model is subject to the capability of the model to establish the necessary references to and from that parameter. This problem is discussed in section 4.3.2, and a solution is offered based on the use of persistent identifiers for all parameters, an object map through which all analysis parameters present in the application can be accessed through their persistent identifiers, and the introduction of a special method, called `setReferences()`, in all the component classes of a model. The mutual referencing of component objects of a model is established, at any time, by calling the method `setReferences()` for each of the component objects. This technique offers a simple, but powerful solution to the problem of assimilating parameters from collaborative sources into a model. However, it implies that the distributed analysis model must be structured as described in chapter 4.

Specification: The distributed analysis model must implement the method `setReferences()`, which establishes the mutual references amongst the parameters comprising an analysis model through use of the parameters' persistent identifiers.

5.3.2 Service methods

Work in the distributed collaboratory is not restricted to synchronous execution, which implies that certain requirements regarding the distribution of parameters can only be provided by constantly running services. Furthermore, centralized administration is needed for services which serve the collaboratory as a whole. Consequently, the services listed below need functional components running on one or more servers (cf section 5.4.4) that are accessible from all of the workplaces:

- **Project analysis parameter database (PAPD) service:** This service makes provision for the storage and retrieval of project analysis parameters to and from the project analysis parameter database (PAPD). This service fulfills a crucial role in the collaboratory, since all official exchange of parameters amongst members of the analysis group takes place through the PAPD. Methods of this service have to meet applicable requirements

regarding the distribution of parameters. Specifically it should support flexible selection of parameters and provide the necessary control over access to parameters.

Specification: The PAPD service has to provide the methods for the storage and retrieval of project analysis parameters, subject to the requirements on the distribution of those parameters. The database technology used in providing the persistent storage is transparent to the clients of this service.

- **Persistent identifier (PID) service:** The analysis parameters of the project are persistently identified. It is the task of the persistent identifier service (PID service) to issue identifiers whenever new parameters are created.

  Specification: The PID service has to provide methods for distributing persistent identifiers for project analysis parameters to the workplaces.

- **Consistency service:** Parameters may be checked out of the PAPD in consistent mode. If a change is made to a parameter used in consistent mode, the change should propagate to other users of the parameter. The consistency service performs this task.

  Specification: The consistency service has to provide methods for propagating changes made to analysis parameters used in consistent mode.

- **Versioning service:** The project analysis parameters are versioned to allow for long duration checkout of parameters without locking access to other users. The versioning service supports updating of parameters' versions.

  Specification: The versioning service has to provide methods for updating of versions of analysis parameters which are in use at any of the workplaces.

5.3.3 Methods at the workplaces

Standalone capability: One of the basic criteria by which the success of any proposed solution to the distributed analysis process would be measured was that of standalone capability, i.e. that work can continue at a workplace in spite of temporary loss of network connection[1]. This implies that the methods used at the workplaces have to be stored in local file systems at the workplaces, and that methods which depend on the services of the collaboratory are capable of buffering communication with those services until a connection can be re-established.

Specification: All methods required for analysis purposes are stored in local file systems at the respective workplaces. Communication to and from the collaboratory services are buffered until connections are established between the services and the workplace.

Phases of the analysis task were described in section 5.3.2. The requirements imposed by the distributed collaboratory on the activities comprising the respective phases are analysed below.

[1] see section 1.2.3

5.3.3.1 Modeling methods

The principal activity of the modeling phase is the creation of the component sets of the finite element model under consideration. In the distributed collaboratory, elements of a model's component sets are either existing project analysis parameters, or new elements of the set of project analysis parameters. Methods employed during the modeling phase have to support the retrieval, assimilation, and modification of existing parameters, and the creation of new parameters which are persistently identified within the project. Methods used in these tasks have to interact with four services of the collaboratory:

- ***Project analysis parameter database service:*** Collaboration amongst members of the analysis group manifests itself in the mutual exchange of project analysis parameters. All exchanges take place through the project analysis parameter database. The PAPD service provides the methods for storage and retrieval of project analysis parameters.
  Specification: The modeling phase requires methods to retrieve parameters, created by a collaborator, from the project analysis parameter database, and to make parameters available for use by others, by storing them in the database. These methods communicate with the PAPD service, and should provide flexible criteria for the selection of parameters for storage or retrieval.
- ***Persistent identifier service:*** All analysis parameters of the project are persistently identified[1]. New parameters, instantiated during the modeling phase, require identifiers issued by the persistent identifier service. During modeling large numbers of new parameters may be instantiated in interactive sessions. Slowing down the modeling task due to latency, or breakdown, in the communication with the PID service have to be avoided. Techniques like the use of temporary identifiers (which are superceded at a later stage), or local cacheing of persistent identifiers have to be supported by the modeling methods. Alternatively the PID service itself may be configured centrally for a project, and then distributed to the workplaces.
  Specification: Methods of the modeling phase have to retrieve persistent identifiers from the PID service for the creation of new project analysis parameters. Tolerance for loss of communication, or slow communication to the PID service should be provided
- ***Consistency service:*** Existing parameters may be retrieved from the project analysis parameter database for use in consistent mode[2]. Any changes made to such a parameter at any of the workplaces propagate via the consistency service to the other workplaces where the parameter is in consistent use.
  Specification: Fundamental changes made to consistent parameters at a workplace have to be forwarded through the consistency service to other consistent users of the parameter. Similarly, changes reported from other workplaces via the consistency service have to be dealt with as part of the modeling task. Methods used in the modeling phase have to communicate with the consistency service to support these actions.

[1] see section 5.2.7
[2] see section 5.2.2

- ***Versioning service:*** The analysis parameters are versioned[1]. During the modeling phase fundamental changes are made to parameters, which require new versions to be issued.
  Specification: Communication between the versioning service and the modeling methods are required for the retrieval of versions for fundamentally changed parameters.

5.3.3.2 Algorithmic methods

Methods of the algorithmic phase assemble and solve the system equations. Large volumes of data are created in the process[2], and the computational requirements can be substantial.

Execution at workplaces: For a broad class of engineering problems, workstations found in engineering offices have the storage and computational capacity required in this phase. In these cases the algorithms could be executed at the workplaces, and the Cholesky factor and primal and dual solution vectors of the `Equation` would be stored in local file systems. It has been argued that the primal and dual solution vectors might be of interrest to other members of the collaboratory, and that their data volumes does not prohibit transferring them to the project analysis parameter database[3].
Specification: The `Equation` parameter should allow for the storage and retrieval of its primal and dual solution vectors in the project analysis parameter database.

Execution on a server: In special cases the size of models which are analysed, or the specialized nature of the analysis algorithms and their computer implementation, may require that they be executed on a particular server. Under such circumstances one would proceed with the modeling phase at the workplaces, use a file transfer protocol to transfer the complete model to the analysis server, perform the solution on the server, retrieve the primal and dual solution vectors from the analysis server, and proceed as if the execution had taken place at the workplace. Consequently no special specification need to deal with this scenario.

5.3.3.3 Interpretation methods

Interpretation of models and analysis results take place in interactive worksessions. Since parameters are not created or modified during this phase, methods supporting interpretation may execute on workstations at the workplaces exclusive of communication with collaboratory services. Normal exchange of parameters makes it possible for any number of members of the analysis group to retrieve a specific model and view its behaviour. Using online communication, or standard telephone communication, it is also possible for them to coordinate what they are viewing and to discuss the results. However, methods which would allow two or more members of the analysis group to look at the same graphics at different locations,

[1] see section 5.2.6
[2] see section 5.2.9
[3] see section 5.2.9

while placing pointers on the graphics to facilitate discussions, would enhance collaboration considerably.

Specification: Methods supporting the interpretation of models and results are not directly tied to collaboratory services. Support of collaborative interpretation, however, will strengthen the feeling of collaboration amongst members.

5.3.4 Distributed programming model

Project server: The methods of the distributed analysis model may be seperated into two sets, namely:

1. A set of methods required for standard structural analysis at the workplaces. These methods are contained in the component classes of the finite element model, and in the class `Analysis` described in sections 4.4 and 4.5, and are installed at the workplaces W_i, see figure 5-3 below, as specified in section 5.3.3 above.
2. A set of collaboratory service methods which support the distributed analysis process. Certain functional components of the collaboratory services have to deal with the collaboratory as a whole, and have to be available on a continuous basis. These classes should be installed on one or more reliable server-computers, called the project server. A project server supporting the various collaboratory services is shown in figure 6.3. Methods of the collaboratory services, which provide support at the individual workplaces only, are installed on workstations at the workplaces W_i.

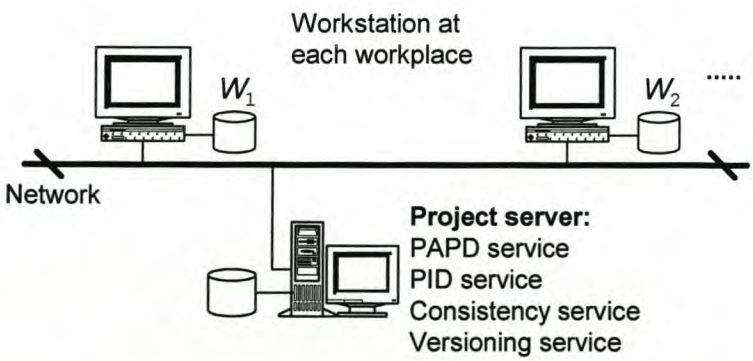


Figure 5-3: Project server of an analysis collaboratory

Specification: For the lifetime of the collaboratory, certain components of the distributed analysis application are installed and available on the project server. The project server should have the required processing and data storage capacity, and have a reliable, high-speed connection to the network.

Network communication: Figure 5-3 depicts a set of computers linked by a network and equipped with the distributed analysis application software. The distributed system accomplishes some of its tasks through network communication between the components at the workplaces and the components on the project server. Such a distributed system can be implemented using a client-server model or a distributed object model. [Mahmoud 2000].

Client-server model: Server-programs act as resource managers with which client-programs communicate for the purpose of exchanging or obtaining information. In the client-server model, communication takes place through a low-level socket connection between the client and the server, and is based on a fixed protocol: the client knows the commands the server expects to receive, the format of the parameters, and the data expected in reply. Using multi-threading, a server can be programmed to serve a number of clients simultaneously. In the structural analysis collaboratory, the number of clients is relatively stable, and typically not very large. Furthermore, the complexity of the communication protocol between the clients and servers can be controlled. As a result the client-server model is well suited for implementing the distributed analysis model. This model is used in the pilot application.

Distributed object model: In this model clients and servers are isolated by encapsulating interfaces [Farley 1998] which describe method signatures. The methods are implemented by the server classes. Software supporting the distributed object model enables the client to obtain a reference to the server object and call the interface methods as if the server resided in the local address space of the client. The distributed object model is very attractive if services should be widely available to an unknown, possibly large number of clients. Within the context of the Internet, this model has become popular. Noteable frameworks supporting the distributed object model are:

- Java Remote Method Invocation (RMI) [Boger 1999]: RMI is a Java-only framework which is part of the core Java application programming interface. Interfaces are defined in Java syntax, and both the client and the server have to be implemented in Java. RMI is attractive if the distributed analysis application is implemented using the Java programming language.
- Common Object Request Broker Architecture (CORBA) [CORBA]: The objective of the Object Management Group (OMG) [OMG] was to define an architecture that would allow heterogeneous environments to communicate at the object level independent of the implementation details of the endpoints. CORBA is an open standard specification, but, since vendors compete on object request broker (ORB) implementations, the envisaged interoperability between different ORB implementations was compromised. This issue was addressed in CORBA 2.0. Interfaces are defined using the Interface Definition Language (IDL), and CORBA objects can be implemented in any programming language that has an IDL mapping. CORBA is a mature and extensive set of standards and has rich implementations [Farley 1998]. For industrial strength implementations a CORBA compliant framework can provide a costly but robust solution.
- Web services [WebServices]: "Web services" is a recent development which exploit existing open standards Internet technologies to provide a distributed computing solution. Specifically, it uses Hyper Text Transfer Protocol (HTTP) as the underlying messaging protocol, eXtensible Markup Language (XML) as the request and response encoding schema, and the Simple Object Access Protocol (SOAP) server environment in which

methods are invoked on objects identified by their Uniform Resource Locator (URL). A significant advantage is that the core technologies mentioned above are open source developments. The promise of cost-effectivity and improved interoperability of different SOAP implementations might make Web services a dominant framework for distributed applications in the future.

The choice of architecture and specific technology for the implementation of the distributed analysis model as a distributed application does not depend on the specific characteristics and requirements of structural analysis. It does, however, depend on cost, available expertise within the company which does the development, and the choice of programming language. Any of the solutions mentioned above will perform adequately if implemented correctly. For the pilot application the lowest level of technology is employed, namely a client-server framework with socket communication.

6 Parameters

The distribution of the project analysis parameters is one of the cornerstones on which the success of the distributed analysis collaboratory depends. For this reason demanding specifications dealing with this issue have been developed in section 5.2. In this section aspects regarding the implementation of the specifications are analysed, and preferred solutions are described.

Specification summary: Core issues of the specifications on the distribution of the project analysis parameters are summarised below:

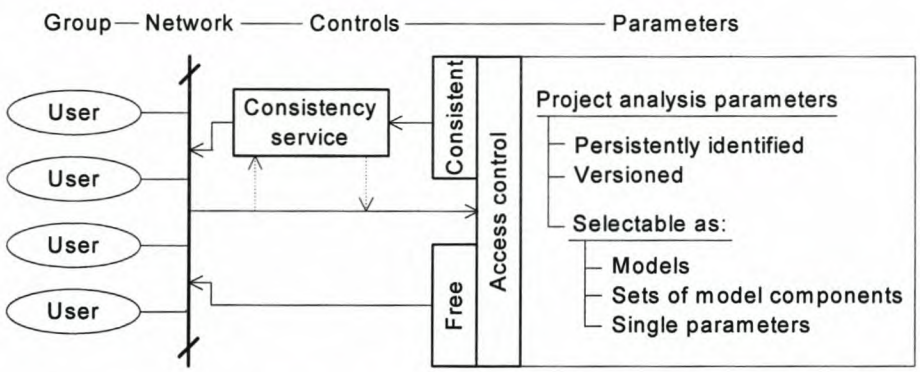


Figure 6-1: Parameter distribution

The members of the analysis group check parameters in and out of the project analysis database, subject to access control. The parameters are persistently identified and versioned, and are singly identifiable and retrievable from the database. Structures built into the database allow the selection of models, as well as the respective sets of components of the model. A parameter may be checked out in free mode or consistent mode. In the latter mode a consistency service is required.

6.1 Persistent identification

Namespace: The analysis application introduced in section 4.3.1 was structured around the use of persistent identifiers. This technique made it possible to refer to an object, which has not yet been constructed in the runtime, as a property of another object. The connection between the objects, as required for execution, is brought about by executing the `setReferences()` method of the object. The distribution of parameters in the analysis collaboratory, and their management in the respective user runtime environments, are also based on the persistent identification of the parameters. Since the analysis collaboratory is formed for the duration of a project, the scope of uniqueness of identifiers (namespace) is the project. In this way sets of parameters can be created project-wide without ambiguity in their identification.

Meaningful identifier: The creation of a unique identifier within a project is in principle not difficult. For example, the sequence of natural numbers provides a unique set of identifiers. However, apart from its position in a sequence, such an identifier posesses no further meaning, and consequently offers very little assistance in navigating to a specific parameter. Assigning a meaningful identifier to each parameter alleviates the navigation problem. The particulars of the project dictate what constitutes a meaningful identifier. Consequently, persistent identifiers should be provided by a service which is specially configured for the project under consideration. A form in which expressive identifiers may be generated is based on a hierarchical scheme:

- Identifier string has the form *A.B.C.D*, in which
  - *A*: Acronym, or index number, associated with the particular task under consideration.
  - *B*: Acronym, or index number, associated with a particular member of the analysis group.
  - *C*: Characters, or index number, indicating the class of the parameter: *MOD (1)* – Model, *EQN (2)* – Equation, *N (3)* – Node, *L (4)* – Local, *M (5)* – Material, *E (6)* – Element, *S (7)* – Support, *NL (8)* – NodeLoad, *LL (9)* – LineLoad, *AL (10)* – AreaLoad
  - *D*: Natural number.
- Example: FLR2.AB.E.15 would indicate the 15^{th} element created by the member with initials AB in the task of analysing FLR2 (floor 2). A shorter identifier string may be obtained if index numbers are used instead of acronyms: 3.9.6.15 would identify the same element, where the member with initials AB has been assigned the member index number 9, and the task FLR2 has been assigned the task index number 3.

6.2 Versioning

Versioning of the project analysis parameters is dictated by the need for long duration checkout of parameters without locking access to these parameters by other users. The problem of proliferation of versions is described, and a specialised user controlled versioning mechanism is proposed as a solution. The selection identifier of a parameter is defined. In this section the problem of consistency of the project information base is not specifically addressed. This is done in section 6.4.

Fullscale versioning: A version of a parameter is a snapshot of its state at a given time. A fullscale versioning mechanism would record each change to each parameter, including changes occurring due to dependencies amongst parameters. The advantage of fullscale versioning is that the complete development history can in principle be reconstructed with the aid of the versioned information. However, the possible proliferation of versions due to dependencies amongst parameters has to be considered.

Direct dependency: Direct dependency between objects is defined as follows [Pahl 2000]:

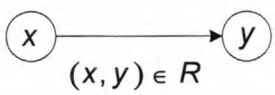
Let M be the object set of an application. An object $y \in M$ is called directly dependent on an object $x \in M$ if at least one of the following conditions is satisfied:

- Methods of object x modify attributes of object y.
- Methods of object y read attributes of object x.

An ordered pair (x, y) of the cartesian product $M \times M$ is called an element of the direct dependency relation R if y depends directly on x:

$$R := \{(x, y) \in M \times M \mid y \text{ depends directly on } x\}$$

If the relation R is visualized as a directed graph, each object of M is shown as a node. The dependency of y on x is represented by an edge which is directed from x to y.



Analysis parameter dependencies:

The possible direct dependencies amongst analysis parameters are shown as a directed graph in figure 6-2. Changing a particular parameter may set off a chain reaction which can be studied by following its effect through the graph[1]. A change to a `Local` parameter, for example, affects the connected `Node`, which in turn affects associated `Element`, `AreaLoad`, `LineLoad` and `Support` parameters. Changes to any of these affects the `Model`, which in turn affects the `Equation`. In a fullscale versioning environment, new versions of

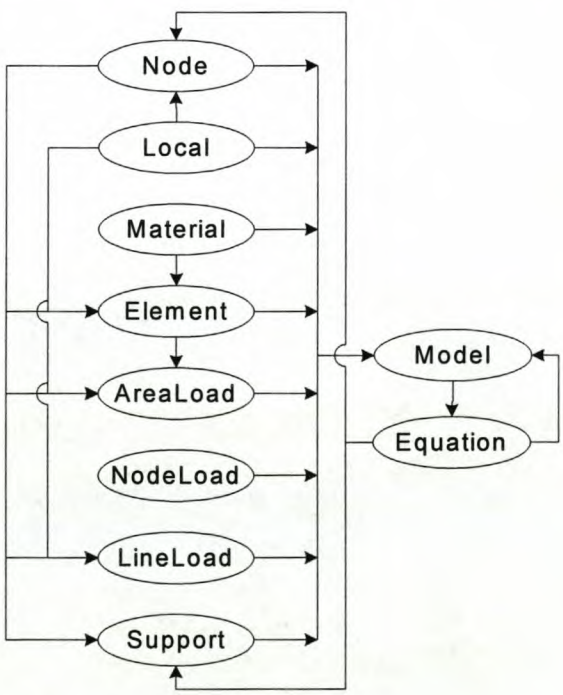


Figure 6-2: Analysis parameters direct dependency graph.

[1] In a more complicated graph the dependencies could be found by computing the transitive hull of the direct dependency relation.

versions of the `Local` itself, and of all the dependent parameters would be required. In spite of the relatively simple structure of dependencies amongst the analysis parameters, a proliferation of versions would result, leading to a complete loss of oversight on the part of the users.

Controlled versioning: In order to avoid the unmanageable growth of versions associated with fullscale versioning, a controlled versioning procedure is developed which takes account of the special properties of the analysis parameters and the work pattern associated with the analysis task: A new version of a parameter is only created if the parameter has been modified fundamentally, and the creation of versions is triggered by explicit user action. The different versions of a parameter are managed in the project base as individual parameters.

Fundamental change: The modification of certain attributes of an analysis parameter represent fundamental changes to the parameter. Changing, for example, the coordinates of a `Node` constitutes a fundamental change to the `Node`, and eventually requires the creation of a new version in order to note the change. A `Node` parameter also has an array attribute in which the displacements of the node are stored when the results are interpreted. All nodal displacements are contained in the primal vector of the `Equation` as well, and are substituted in the nodes in a simple operation. This creates the possibility that the displacements in the `Node` itself may be regarded as transient information, since the model's primal vector can be stored and used to backsubstitute nodal displacements when required. Exploiting this possibility is convenient, since the same `Node` may then be used in a number of models. The same reasoning may be applied to the other analysis parameters, and new versions of parameters are only required if a fundamental change has been effected. Fundamental changes to the analysis parameters are listed in table 6-1. Changes to attributes not mentioned in the table are considered transient.

Parameter	Fundamental change
`Model`	Adding and removing components Fundamentally changing a component
`Equation`	Any change
`Node`	Coordinates, system indices, local system
`Local`	Angle or direction cosines
`Material`	Young's modulus, Poisson's ratio
`Element`	Node identifiers, material identifier, thickness
`NodeLoad`	Node identifier, load intensity
`LineLoad`	Node identifiers, load intensity
`AreaLoad`	Element identifier, load intensity
`Support`	Node identifier, support conditions

Table 6-1: Fundamental changes to analysis parameters

Change propagation: During the modeling phase the engineer makes any number of fundamental changes to the parameters of the model under construction, and often the same parameter is modified more than once. A change to a parameter may affect other parameters through the demand for followup changes, i.e. change propagation requirements. Consider, for example, a change to a `Local` parameter, i.e. the local direction is modified: With the aid of supporting software tools, the engineer should propagate the change so that it is reflected in a `Support`, a `LineLoad` and a `NodeLoad` which are associated with the modified `Local`. The propagated changes are fundamental, and new versions of these parameters are eventually required. However, the new versions need not be created right away, since the same parameter may undergo a number of changes during the modeling phase. Intermediate versions, if they were created, would be helpful in reconstructing the events that took place during modeling, and they might facilitate backtracking. However, they would clutter the project database, and have no use other than that stated above.

Updating: The computation of the attributes of dependent objects due to changes in the objects on which they depend is called an update of the applicaton. An investigation of an analysis update shows that the `Model` and its `Equation` undergo fundamental change during such an update, but the changes in the other classes of component objects are transient. Hence an analysis update may result in the creation of new versions of the `Model` and its `Equation`. However, the actual creation of new versions should be postponed. In many cases the first investigation of results show up the mistakes and shortcomings of the model, resulting in additional changes followed by another analysis update.

User controlled versioning: The appropriate time to create new versions of fundamentally modified parameters depends on the intentions of the engineer. Consequently, new versions, where required, should not be created automatically, or right away. Typically the completion of a model which, in the view of the engineer, yields useful results is the time when the model and its components should be versioned. In this way the number of versions are minimised, and the versioned parameters are associated with meaningful models, thereby reducing the loss of oversight significantly.

Implementation: The implementation of user controlled versioning of the analysis parameters requires two extentions to be made to the parameters introduced in chapter 4:

1. Each project analysis parameter needs a `version` attribute: The `version` attribute is an object which itself has three attributes:
   - `number`: A natural number which is incremented by one each time a new version of the parameter is created.
   - `timestamp`: The time at which the version is created. In case backtracking is required, the time at which snapshots of parameters were taken are helpful in ordering events.

- `history`: The sequence of all version numbers of the parameter. Since two or more members of the analysis group may be using the same parameter at a given time, modify it, and create different new versions, the version numbers of the parameter do not follow the natural number sequence.

2. Each project analysis parameter registers fundamental changes to itself: New versions of parameters are only created if the parameter has been changed fundamentally. In order to postpone the creation of versions to a user controlled point in time, analysis parameters register the occurence of fundamental changes to themselves. Through the implementation of an interface a parameter can be queried about fundamental change at versioning time.

The versioning operation traverses the set of parameters (typically a model) to be versioned. For each parameter which has registered a fundamental change, a new version attribute is created.

Selection identifier: In order to uniquely identify a specific version of a parameter in the project base, the selection identifier (sid) is used. The selection identifier is a combination of the parameter's persistent identifier, which is unique in the project, and version number, which is unique for the parameter. Since the persistent identifier is a string, it is useful to concatenate the persistent identifier and the string representation of the version number. For readability the character ':' is used to seperate the concatenated strings:

Selection identifier: sid = persistent identifier : version_number = $p : v$

For example: FLR2.AB.E.15:2 would indicate version 2 of element 15, the original version of which was created by the member with initials AB in the task FLR2.

After versioning, each modified parameter has a new (and unique) selection identifier (sid). The newly versioned parameter can then be added to the project analysis parameter base (as an individual), and retrieved using its selection identifier as key.

Versioned model: A project analysis parameter of class `Model` comprises a set of versioned component parameters of classes `Equation`, `Node`, `Local`, `Material`, `Element`, `NodeLoad`, `LineLoad`, `AreaLoad` and `Support`. A particular version of a component parameter may belong to any number of models. `Model` parameters are versioned as well.

Consider, for example, version number k of a model with identifier m. This model is uniquely identified by its selection identifier: $m : k$.

If M_k is the set of component parameters of the model, and $\Sigma^{m:k}$ is the set of selection identifiers of these parameters, then:

$$M_k = Q_{m:k} \cup N_{m:k} \cup L_{m:k} \cup T_{m:k} \cup E_{m:k} \cup D_{m:k} \cup F_{m:k} \cup A_{m:k} \cup S_{m:k}$$

$$\Sigma^{m:k} = \Sigma_Q^{m:k} \cup \Sigma_N^{m:k} \cup \Sigma_L^{m:k} \cup \Sigma_T^{m:k} \cup \Sigma_E^{m:k} \cup \Sigma_D^{m:k} \cup \Sigma_F^{m:k} \cup \Sigma_A^{m:k} \cup \Sigma_S^{m:k}$$

in which the sets containing component parameters and their selection identifiers are:

$Q_{m:k}$	equation in model $m:k$	$\Sigma_Q^{m:k}$	selection identifier of the equation $Q_{m:k}$
$N_{m:k}$	set of nodes in model $m:k$	$\Sigma_N^{m:k}$	selection identifiers of nodes in $N_{m:k}$
$L_{m:k}$	set of locals in model $m:k$	$\Sigma_L^{m:k}$	selection identifiers of locals in $L_{m:k}$
$T_{m:k}$	set of materials in model $m:k$	$\Sigma_T^{m:k}$	selection identifiers of materials in $T_{m:k}$
$E_{m:k}$	set of elements in model $m:k$	$\Sigma_E^{m:k}$	selection identifiers of elements in $E_{m:k}$
$D_{m:k}$	set of nodeloads in model $m:k$	$\Sigma_D^{m:k}$	selection identifiers of nodeloads in $D_{m:k}$
$F_{m:k}$	set of lineloads in model $m:k$	$\Sigma_F^{m:k}$	selection identifiers of lineloads in $F_{m:k}$
$A_{m:k}$	set of arealoads in model $m:k$	$\Sigma_A^{m:k}$	selection identifiers of arealoads in $A_{m:k}$
$S_{m:k}$	set of supports in model $m:k$	$\Sigma_S^{m:k}$	selection identifiers of supports in $S_{m:k}$

The different selection identifier sets are stored with the model in the project database, thereby providing access to the natural sets contained in the model, e.g. set $\Sigma_N^{m:k}$ can be used to retrieve the nodes associated with version k of model m.

6.3 Access control

Operating system controls: Standard access control mechanisms provided by operating systems are login control, control over access to branches of the filesystem tree, and control over access to individual files. These controls are useful in filtering down system users to members of the analysis group for global access to the project analysis information. However, in the distributed collaboratory access to subsets of the project analysis parameters are required.

Control granularity: It is envisaged that complete models and the sets of component parameters of a model will be the standard units which are processed and exchanged in the collaboratory. However, access to individual parameters is also possible, and sometimes necessary, which implies that access control at the individual parameter level is essential.

Access mode: Parameters can be used in consistent mode or in free mode, as explained in section 4.3.2. These two access modes require different access controls. If a parameter is used in consistent mode, modifications have a drastic influence on existing models and other concurrent users. As a result consistent mode users should be in the position to judge on the necessity of modifying a parameter. Free mode usage is provided as a convenience, and does not impact on existing models and other users. Consequently looser access control could be allowed for free mode usage.

Identification: Each member of the analysis group has two identifiers, a unique personal identifier, and the identifier of the taskgroup to which he belongs. A taskgroup is a subgroup of the analysis group which collaborate on a specific tasks. Positive personal identification is required for any form of access to the project information base. A positively identified member may add new parameters to the project base. Checkout of a parameter, however, is subject to access rights listed on the parameter's access tag.

Access tag: Each parameter has an access tag containing the personal identification of its owner, its taskgroup identification, and rights for the two checkout modes pertaining to the owner, a taskgroup member, and an analysis group member:

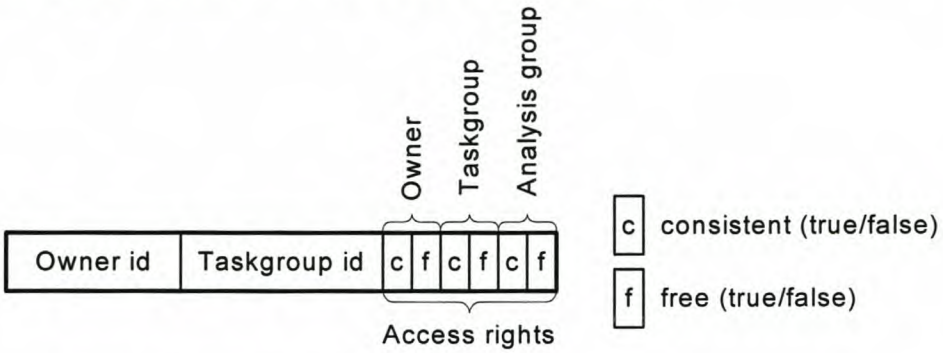


Figure 6-3: Parameter access tag

The creator of a parameter, or a new version of a parameter, is its original owner. The owner sets the taskgroup identifier and the access rights, and may also appoint a new owner. Only the owner of a parameter, and the analysis administrator, can edit the parameter's access tag. Users can setup default access tags. Typically the analysis group should be allowed free mode access to parameters, while owners and taskgroup members have free and consistent mode access.

6.4 Consistency

Service and control: Structural analysis is a technically complex task, and many factors are taken into account when decisions are made. In the analysis collaboratory, the decisions of a member processing certain common parameters may have a big impact on existing models, and the work of other members. A single change to a parameter may invalidate the computed results of a model, and recomputation may involve hours of processing time. Consequently strict control of consistency is not considered to be a suitable mechanism in a structural analysis collaboratory. Experience has also shown that too much control stifles the collaboration process. [Borghoff 2000] lists a number of concepts, which were implemented in the design of a successful product supporting collaboration in a shared information space. Particularly relevant are:

- Overwriting of existing information, and hard consistency requirements are avoided. New entries are simply appended.
- Users recognize conflicting replicas easily and can solve the conflicts manually.

The collaborative structural analysis environment requires a consistency service, which

- advises concerned members of the analysis group of possible consistency problems, and
- provides the neccessary information to pursue the matter, if required.

Cases of a model: A finite element `Model` is constructed using parameters which describe

- geometry (`Node, Element, Local`),
- material (`Material`),
- kinematic boundary conditions (`Support`), and
- applied loading (`NodeLoad, LineLoad, AreaLoad`).

During the algorithmic phase the model's `Equation` is constructed and solved. The attributes of an equation, and the parameters which influence these attributes are shown in figure 6-4. Once the Cholesky factor of the matrix has been computed, new results due to changes in the applied loading can be determined without much effort, since only the system load vector $\mathbf{q}_s$ needs to be recomputed followed by the forward and backward sweep to compute the results. This is an intrinsic property of linear structural analysis, and led to the introduction of the case concept for loads. This concept is extended for geometry, material and kinematic boundary conditions (support). Changes to the geometric parameters of a model lead to a new geometry case of the model, and a new material case or support case results from respective changes to the materials or support conditions of the model. The different geometry, material and support cases, however, are associated with different stiffness matrices of the model, and consequently require significantly more computational effort than a new load case.

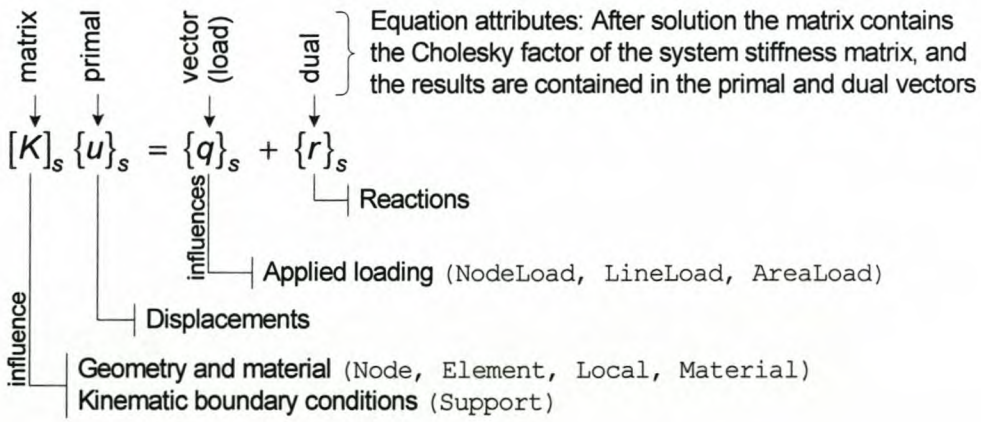


Figure 6-4: Equation attributes and parameter influences

Consistency mechanism: The various models which are developed and used in the analysis of a built facility each comprise a number of parameters. Within an analysis collaboratory, the exchange of parameters creates the potential for certain parameters to be deployed in more than one model. A parameter is said to be consistent in certain models if its fundamental state, i.e. its version, is the same in those models. When a parameter is checked out of the project database in consistent mode, the purpose of the consistency service is to ensure that any fundamental change to the parameter is reported to other models in which it is used. However, it is considered the duty of the owner of a model to act on reported changes.

The affected models may either be in a dormant state in the project database, or in dynamic use (i.e. under construction) at one or more of the workplaces.

- *Dormant model:* A specific version of a project analysis parameter is uniquely identified within the project, and hence within each model, by its selection identifier (sid). If a parameter `sid-old` is checked out of the project database in consistent mode and modified, a new consistent mode version `sid-new` is eventually checked into the project database. The `old` and `new` consistent mode version numbers of the parameter are registered in the database. As a result, the consistent mode evolution of parameters is known. When a model which contains `sid-old` is checked-out of the database, the user is informed that later consistent mode versions of the parameter are available.
- *Dynamic model:* Members of the analysis group check parameters out of the project database for further processing. For each group member a checked-out parameter either belongs to his consistent set, or to his free set, depending on the mode in which he requested the parameter. Changes to parameters checked out in free mode are not reported to other members. However, if any member makes a fundamental change to a parameter belonging to his consistent set, the change is immediately propagated to other members that are using the same parameter in consistent mode. Similarly, when a new version of a consistent mode parameter is created, the new version is propagated to the other consistent mode users of the parameter. Further action is the responsibility of the member being notified. While a parameter is checked out, changes to it are accumulated. The accumulated changes are reported to a member who checks out the parameter at a later point in time.

Procedures: The consistency mechanism described above requires four distinct procedures, which correspond to those member actions which influence others in the analysis group. These are the member actions of checking-out, modifying, creating a new version, and checking-in a parameter. The procedures are presented in the form of flow diagrams below. The diagrams refer to terminology which needs to be defined:

- Consistent representative: Each parameter that is checked-out in consistent mode has a representative which maintains its list of consistent users and its list of modifications. The consistent representatives are managed by the consistency service in a central location.
- ConsistentUserList: The list, maintained by the consistent representative of the corresponding parameter, of users that have checked-out the parameter for use in consistent mode.
- ModificationList: The list, maintained by the consistent representative of the corresponding parameter, of all fundamental changes made to the parameter by its consistent users. This list includes new versions of the parameter.

Checking-out a parameter: If the user has the appropriate access rights he receives the parameter. For a consistent-use parameter its consistent representative is created if it does not already exist, and the user is added to its consistentUserList. The parameter's consistent-use flag is set to according to the check-out mode. This makes consistent parameters aware that they should report modifications. After a consistent parameter is checked out, all accrued changes to it are reported, as well as new changes when they are made.

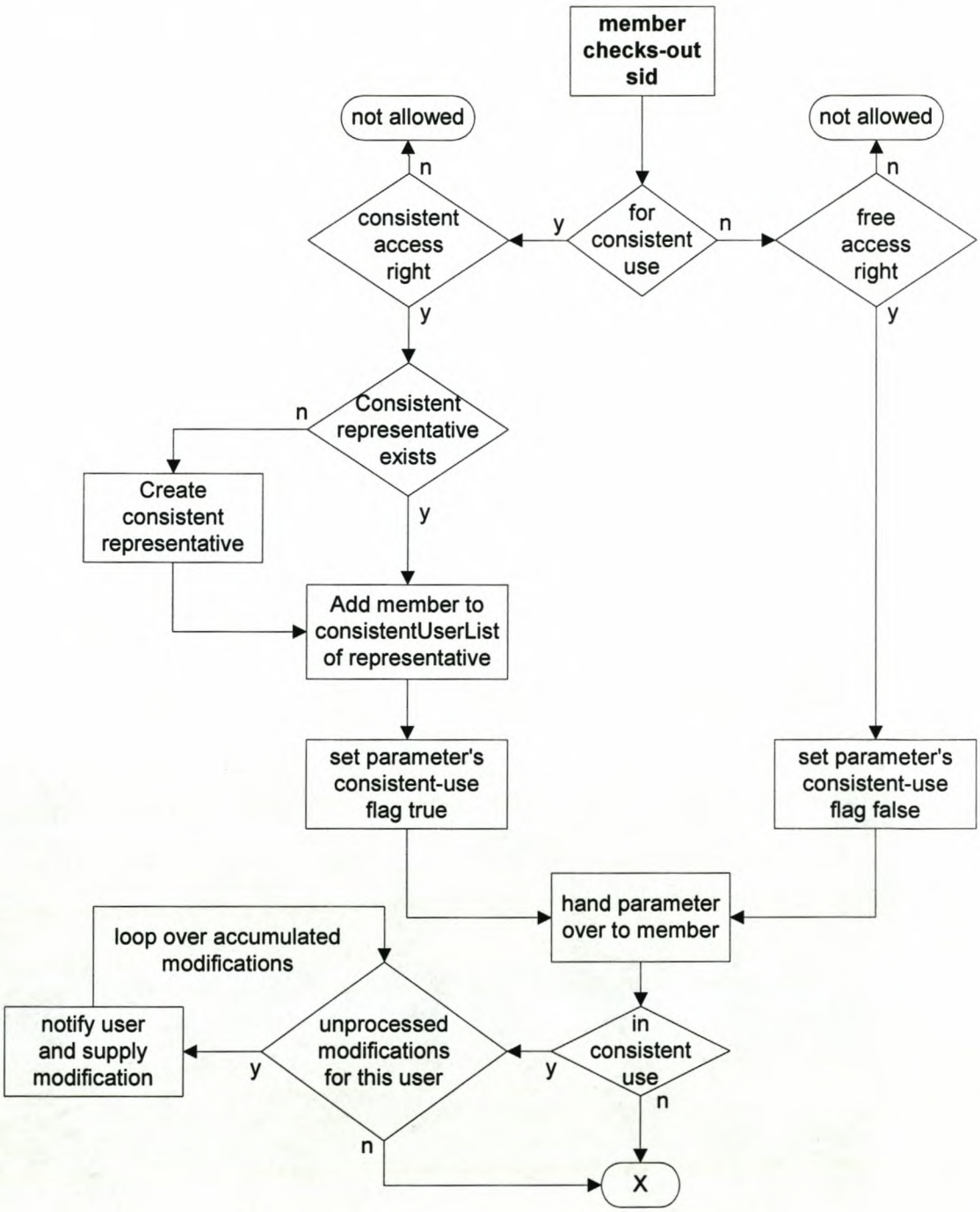


Figure 6-5: Parameter check-out procedure

Modifying a parameter used in consistent mode: A fundamental change to a parameter used in consistent mode is propagated by the consistency service. Each modification is noted in the modificationList of the parameter's consistent representative, and forwarded to other concurrent users of the parameter who may choose to implement the change or to ignore it.

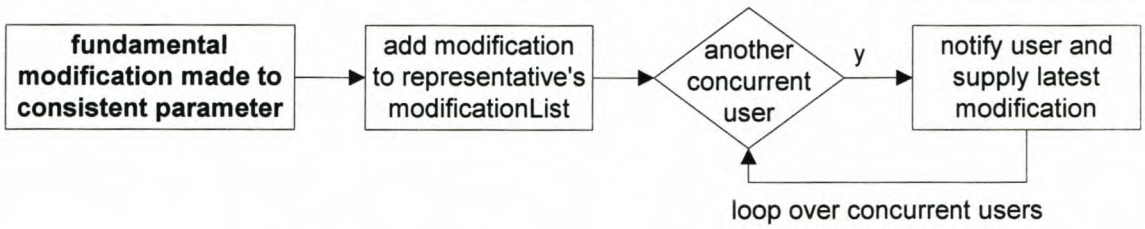


Figure 6-6: Consistent parameter modification

Creating a new version of a parameter used in consistent mode:
If a user creates a new version of a consistent parameter, a consistent representative is created for the new parameter. The user is added to the new representative's consistentUserList, and removed from that of the previous representative. Similar changes are made for other concurrent users that choose to accept the new version. Some may choose not to accept the new version due to differences dictated by their particular circumstances.

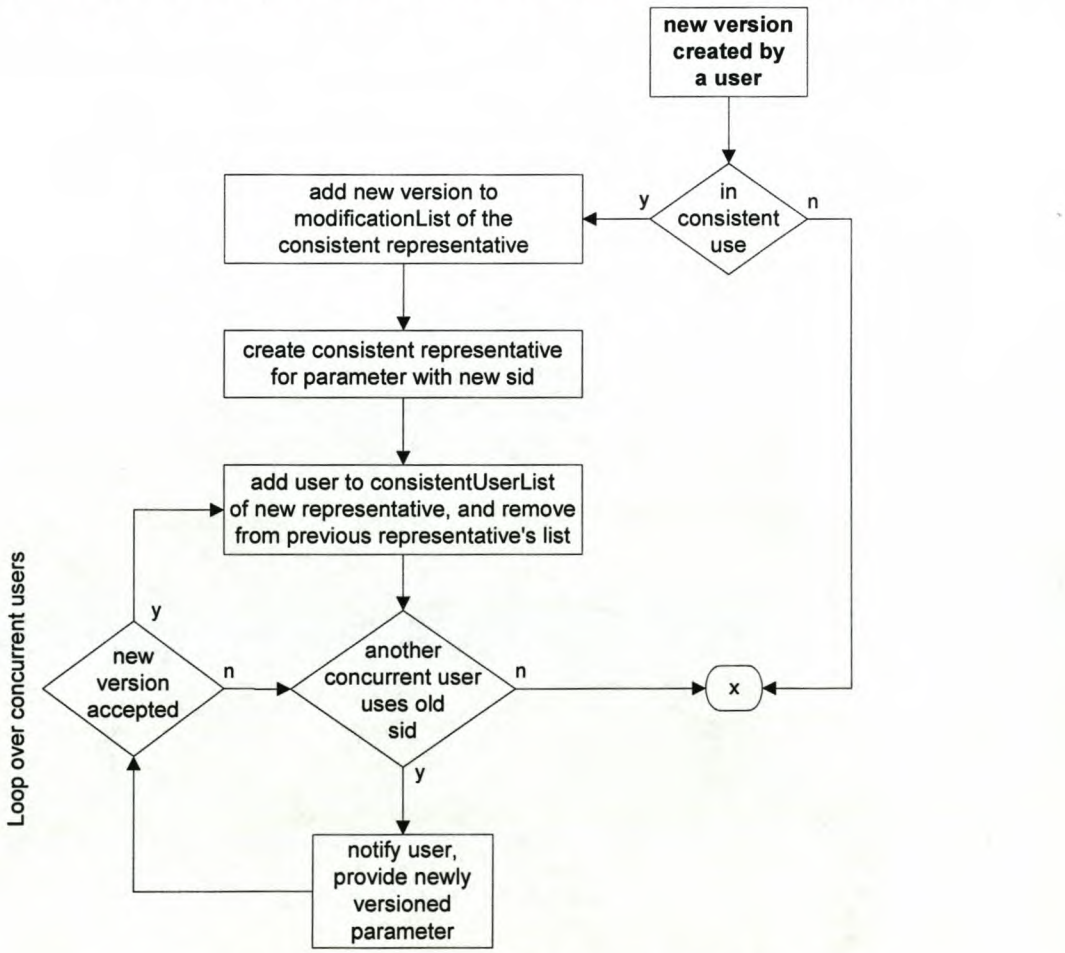


Figure 6-7: Creating a new version of a consistent parameter

Checking-in a parameter: At an appropriate point in time a user checks parameters into the project database. New parameters are always added to the database. The user that checks in a parameter he had used in consistent mode is removed from the consistentUserList of the parameter's representative.

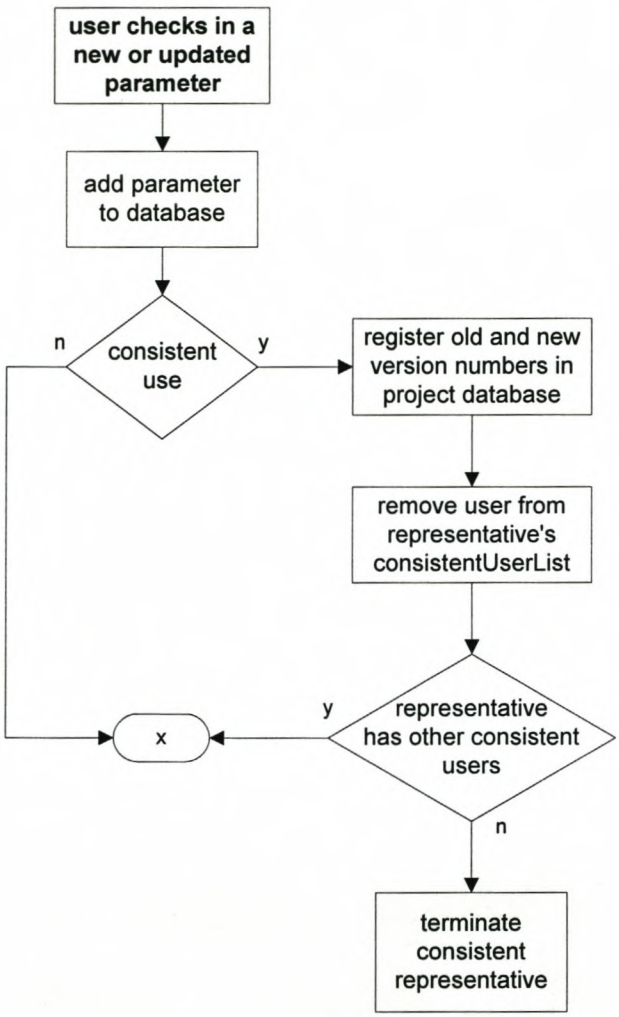


Figure 6-8: Parameter check-in procedure

6.5 Persistence

The physical storage location of the project analysis parameters, and the database technology which is to be employed are under investigation in this section. Alternatives are presented and evaluated in terms of the specifications developed in section 5.2. Preferred solutions are proposed.

6.5.1 Storage location

Collaboratory architecture: The basic architecture of the collaboratory is shown in figure 6-9. It is assumed that all the filesystems at the workplaces can be reached and utilised for the storage of project parameters.

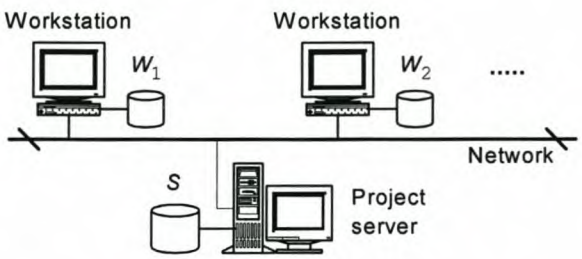


Figure 6-9: Collaboratory architecture

As extreme cases of storage location, all parameters may either be stored on the project server, or alternatively on the workstations of the analysis group.

- **Monolithic storage on server:** If P is the set of all project analysis parameters, and S is the set of analysis parameters stored on the project server, then $S = P$, i.e. all project analysis parameters are stored in a central database on the server. Significant advantages of this configuration are that information is controled and managed in a central environment, which is the same for all members of the group, and redundant storage is avoided. However, the server represents a single point of failure and, since no parameters are available at the workplaces when connection to the server is lost, standalone capability at the workplaces is not supported. Furthermore, since processing takes place on the workstations, repeated transfer of parameters between the workplaces and the server is subject to the latency associated with network communication at each transfer.
- **Distributed storage at the workplaces:** The sets (W_k) of analysis parameters stored at the respective workplaces are a decomposition of the set of all project analysis parameters (P):

  $$P = W_1 \cup W_2 \cup \ldots \cup W_n \qquad \text{and} \qquad W_1 \cap W_2 \cap \ldots \cap W_n = \{\ \}$$

  in which n is the number of workplaces in the collaboratory.

  The set of parameters stored at workplace k, W_k, would typically be the parameters which were instantiated at k. The distributed sets have to be tied into a logical database unit by management facilities installed on the project server, with the result that the

server still represents a single point of failure. Standalone capability at the workplaces is not fully supported, since parameters stored at one or more of the other workplaces may be used in a local model. If the server fails, or the connection to one of the other workplaces is lost, all the parameters of the local model are not available. This configuration is an improvement on server-only storage with regards to its dependency on network communication, since local parameters are excluded from network transfers.

Preferred storage configuration: The preferred storage configuration is dictated by the requirement of standalone capability at the workplaces. Processing of local models at each workplace, in spite of possible network or server failure, demands local storage of all parameters used in local models. By regarding these local storage sets as duplicates, the advantages of both the central and distributed storage configurations can be exploited:

- **Parameters used in local processing are duplicated at the workplaces:** All parameters used in an analysis task are stored in a filesystem at the workplace. Since parameters are shared in the collaboratory, the shared parameters are stored at more than one workplace, i.e. the local storage sets, $W_k \quad k = 1,\ldots,n$, are not a decomposition of the project parameter set P. Maintaining the consistency of the parameter base is a problem in such a configuration which contains redundant entries. This problem is avoided by excluding locally stored parameters from the project parameter base which is managed by the distributed analysis model, although members are informed of changes to parameters used in consistent mode[1]. As a result, the local storage sets serve only local purposes:
  - Standalone capability at each workplace is supported through the availability of all parameters used in local models.
  - Intermittent local storage preserves the state of the model under consideration in the case of a system or application failure.

  The management of the local storage sets is the responsibility of the individual members of the analysis group, and is similar to such management in non-collaborative tasks.
- **The complete set of project analysis parameters is stored in a monolithic database on the project server:** Under the management of the distributed analysis model, all analysis parameters are stored in, retrieved from, and exchanged through a monolithic database which resides on the project server. The advantages of central storage and management are achieved at the cost of redundant storage at the workplaces. This redundant storage, however, has the advantage that network-based transfer of parameters between the central database and the workplaces can be kept to a minimum, since storage and retrieval between worksessions involve only local actions.

[1] see section 6.4

6.5.2 Database technology

Database technology options: Persistent storage of objects is the subject of much debate, and a "best" way of providing object persistence is not on hand. A number of storage and database technologies are available. Three alternatives, namely serialization, object oriented, and relational (including object-relational) database technology are briefly described and evaluated in terms of the requirements of the project analysis parameter database (PAPD).

6.5.2.1 Serialization

Object state: The state of an object is the set of instantaneous values of all its attributes. Objects may inherit attributes from superclasses, and reference other objects as attributes. Consequently an object's state depends on the values of its primitive attributes and on the states of the objects it references. Values are represented in the computer as sequences of byte-states in computer memory. Therefore the state of an object can be persistently stored by writing the binary representations of the complete object graph to a disk file.

Java serialization: The Java serialization mechanism converts an object graph into a byte stream called the serialization stream. It extends the concept mentioned above by assigning a serial number to each object which is written into the stream. If an object is referenced by more than one object in the stream, the referenced object is physically placed in the stream only on the first occasion, and its serial reference (also called the back reference) is substituted in following references to the object. De-serialization is the reverse process in which objects present in the stream are instantiated and their attributes are assigned the values that were encoded in the stream.[Li 1998]

Evaluation: Serialization was designed for lightweight persistence and to encode objects into byte-streams for network transmission. It provides a compact storage mechanism, but it has the disadvantage that a single object cannot be retrieved from a serialized stream; the complete stream has to be de-serialized. Furthermore, schema evolution is supported very weakly; changes to any class of which instances are contained in a serialized stream might make the stream unreadable. Also problematic, from the point of view of the requirements of the project analysis parameter database, is the fact that serialization does not provide any database utilities (e.g. a query language). These shortcomings preclude the use of standard serialization as a technology for establishing the PAPD. However, the disadvantages of serialization does not apply if it is used to store of the state of a model under consideriation at one of the workplaces. The purpose of local storage is to save and restore the complete state of a model under consideration between worksessions, to serve as a backup mechanism in the case of a local failure, and to provide a standalone capability in the case where access to the PAPD is lost. For these purposes serialization offers an easily implemented alternative.

6.5.2.2 Object oriented database

Object oriented paradigm: The modeling power of the object oriented programming paradigm encouraged the development of a database technology which is compatible with the object and data structures used in the programming models. With classical database technology there is a clear distinction between the database, which contains passive data, and the application, which contains the program logic. In object oriented technology this distinction vanishes, because the database stores objects which may contain both data and methods.

Advantages: Object oriented database technology is specifically aimed at supporting the programming paradigm, and provides a direct way of implementing object persistence. The persistence service is tightly integrated with the application, which leads to a simpler system architecture in which program variables directly (for the programmer transparently) reference persistent objects. See Figure 6-10. This efficient interface, based on the the reference semantics of object oriented models, leads to good runtime performance, even when operations are frequently executed on individual, or sets of persistent objects.

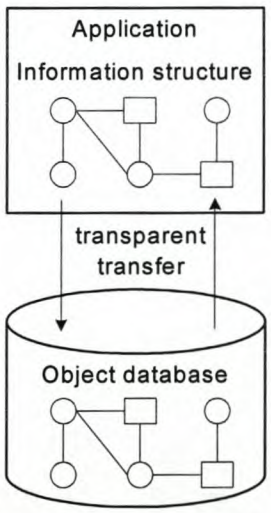


Figure 6-10: Architecture with OO database

Disadvantages: The tight integration between the application and the database is also the cause of a serious problem, namely that changes to the application classes lead to incompatibility with existing databases, i.e. the schema evolution problem. [Hansen 1999] and [Goodman 1995] describe this problem, as well as difficulties regarding the administration and management of object oriented databases in multi-user environments. Furthermore, a serious disadvantage associated with the use of an object database for the role of the PAPD, is the lack of standardisation with regard to an object-database query language. Although any required functionality can be established programmatically in the database, it is allways limited to the specific functionalities which have been foreseen by the programmer.

The extention of functionality requires a large and specialised programming effort, which is very undesirable in the dynamic environment of a structural analysis collaboratory.

Evaluation: Utilisation of an object oriented database is indispensable when the object model is complex and the application frequently operates on the persistent objects. For example, the local storage sets (W_k, section 6.5.1) may profitably be realised using object databanks since these allow the persistent state of a model being processed to be updated continuously in the local database.

The specifications which specifically relate to the project analysis parameter database[1] (PAPD), however, are based on considerations regarding the exchange of information in the collaboratory, not the processing thereof. Parameters are selectively checked-out of the PAPD by a user, processed in a separate runtime environment, and checked-in to the PAPD at a later point in time. The check-out, check-in actions are relatively low-frequency operations, but may require complex and variable selection and access control procedures to be executed on large volumes of data. Furthermore, the application introduced in chapter 4 is structured around the use of persistent identifiers to model relations between objects, which reduces the complexity of the analysis parameters dramatically. The strong point of object oriented database technology, namely its capacity for high-frequency processing involving complex persistent objects, is not required for the project analysis parameter database. Its main disadvantages, however, i.e. its lack of a flexible and standardised query language and difficulties with schema evolution, undermines its suitability as a preferred technology for the PAPD.

6.5.2.3 Relational database

Relational model: Relational database technology is based on the relational model of data. A database comprises a set of relations, each relation being a set of elements. The elements are identified by keys, and their structure is defined by attributes whose respective values fall inside given domains. The relations are represented as tables with rows and columns, in which each row contains an element of the relation, while the element's attributes are stored in the columns. Relations between elements are not based on references, but on identifying a referenced element by means of its key[2].

Advantages: Relational database technology has been developed and used for a relatively long time, and as a result implementations have become robust and fast. The success of the technology has its roots in the extreme simplicity of the relational data model, which requires a single datastructure: tables with rows and columns containing scalar datatypes. Amongst the current database technologies, relational databases are the most widely used. A considerable advantage associated with relational technology is the fact that almost all implemen-

[1] see section 5.2.10
[2] Note the similarity with the application structure, introduced in chapter 4, based on the use of persistent identifiers to realise the reference-connection between objects in the runtime.

tations support a standardised and powerful query language: the Structured Query Language ISO-ANSI SQL. SQL is based on relatively few, well defined operations which have a strong mathematical basis. Furthermore, most relational database managing software implements the interfaces ODBC (Open Database Connectivity) and JDBC (Java Database Connectivity), thereby providing database services (SQL) to program developers.

Disadvantages: Two predominant issues may impact negatively on applications using relational database technology to provide object persistence:

- *Impedance mismatch:* The term refers to a mismatch between the frequency and complexity of operations performed in an application, and the frequency and complexity of operations which relational databases are well equipped to handle. Object oriented applications execute relatively simple operations at a high frequency, while relational databanks are more equipped to execute complex operations on large data sets, at a much lower frequency. The way in which an application uses the persistence service determines whether the impedance mismatch is a major problem.

Figure 6-11: Impedance mismatch

- *Relational model ↔ Object model conversion:* The relational model of data is not compatible with the object oriented paradigm, leading to a gap between the application model and the database model which has to be overcome by the programmer [Boger 1999]. A seperate persistence tier has to be added to the program architecture, since the transparent use of persistent objects provided by the object databases falls away and the distinction between passive data in the database tables, and program logic (objects) in the application becomes clear. In effect, an object has to be taken apart to be stored in the database, then assembled again upon retrieval. If the object model contains complex relations, this issue becomes a dominant problem, and using an object oriented database is the better choice.

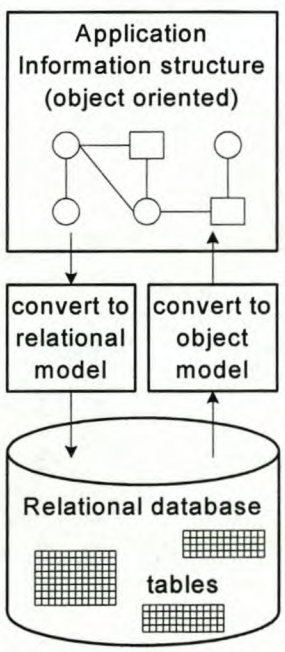


Figure 6-12: Architecture with relational database

Evaluation: The role and usage pattern of the PAPD determine whether the predominant problems of using relational database technology in an object oriented processing environment, described above, actually represent serious drawbacks:

- *Impedance mismatch:* The PAPD is accessed by the members of the analysis group to store parameters as dictated by their obligations as set out in the collaborative analysis schedule, and to retrieve parameters from collaborative sources for further processing. These actions may require careful and detailed selection procedures, which is one of the strong points of relational databanks, but are executed at a relative low frequency, which does not activate the impedance mismatch described above.
- *Relational model ↔ Object model conversion:* As explained above, the use of relational storage technology introduces a third tier into the application architecture:

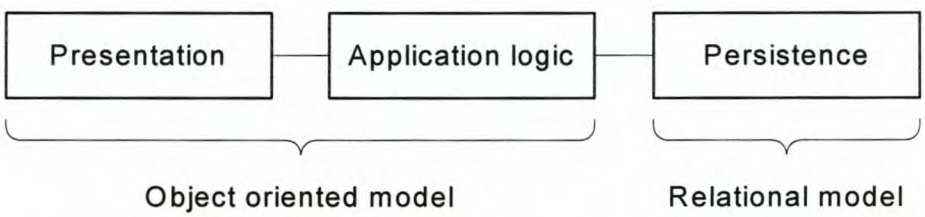


Figure 6-13: Three-tier architecture

The storage of object models with complex relations in a relational database is problematic. In general, however, object persistence based on relational technology is appropriate "when objects are heavy with data and light on relationships" [Morgan 1999]. The structure of the finite element analysis application introduced in section 4.3.1 is based on the use of persistent identifiers to model relations between objects. This means that the application deals with the object relations itself, and the relations need not be mapped to the persistent model. As a result the project analysis parameters can be stored in a relational database with ease. Furthermore, the processing of the parameters takes place at the distributed workplaces, which implies that parameters need to be transferred from the PAPD, which resides on the server, to the member workstations via the network. Network transfer of an object implies that the object has to be serialised. If the PAPD is a relational database, an object need not first be constructed on the server side, and then serialised for transfer. Data coming out of the database can be encoded directly and placed in the transfer stream.

The suitability of relational technology with respect to the specific requirements described in section 5.2.10 are considered:

- *Flexibility, media-suitability, navigability:* It has been argued above that the structure of the analysis application around the use of persistent external identifiers greatly reduces

the complexity of the object model and creates the opportunity to easily map the analysis parameter information to a relational persistent model. The simplicity of the relational model allows the query language (SQL) to provide user friendly constructs for the descriptive and flexible formulation of database queries. SQL provides extensive support for finding and selection of data, e.g. the *select-from-where* construct, which can be used to provide certain navigation aids and selection procedures (based on structural engineering semantics) at the user surface. For non-standard queries, the full power of raw SQL could be made availabe.

- *Data volume:* Relational databases scale well, and remain performant in the face of large data volumes. [Boger 1999]
- *Extended use:* The use of the project analysis parameter database for purposes other than originally intended during the analysis phase is enhanced by the use of a relational database. SQL is a standardised language, which implies that utilities for extended usage can be developed without undue effort.

The use of a relational database to realise the project analysis parameter database (PAPD) is not subject to the typical disadvantages associated with the use of relational technology to provide object persistence. Furthermore, the simplicity of the relational model, and the standardised query language (SQL) support the intended usage of the PAPD in a flexible way. As a result, relational database technology is considered an appropriate choice for the PAPD, and is used in the pilot implementation.

7 Methods

The methods of the distributed analysis model determine its functionality, and through that the capacity of members of the analysis group to work efficiently. Methods which support structural analysis in the local environment were described in sections 4.4 and 4.5. In order to satisfy the standalone specification[1], the classes providing these methods are installed on local file systems at the workplaces, and function as described. In this chapter the extentions required to support the distributed analysis process are explored in the light of the specifications developed in section 5.3 for the distribution of methods. Where applicable, alternative implementation techniques are investigated.

Activity summary: In the analysis collaboratory, members of the analysis group create, process and interpret the results of finite element models described by parameters which are shared amongst the members. To support collaborative sharing of parameters, the parameters need to be:

- persistently identified with the aid of the persistent identifier service,
- versioned with the aid of the versioning service,
- checked in and out of the project analysis parameter database and distributed from there to other members of the analysis group with the aid of the project analysis parameter database service,
- usable in free mode or consistent mode, the latter case being supported by the consistency service.

The above mentioned collaboratory services are distributed applications with functionality at the workplaces supported by methods on the project server.

7.1 Computing platform

The group responsible for the structural analysis task of a project is formed ad hoc, and the computing platforms used by members of the group may differ significantly, both in hardware and in operating environment. Due to the standalone criterium, i.e. that work can continue at a workplace in spite of a temporary loss of network connection, the use of Internet browsers as an operating environment is not possible. Consequently the distributed analysis application has to provide the required functionality in a setting of diverse computing platforms. This implies that the application has to execute on all possible platforms which may occur in an analysis collaboratory. In this effort the choice of implementation language plays a decisive role.

[1] see section 5.3.3

Porting: The fundamental requirement pertaining to the choice of a language for implementing the distributed analysis application is that the application should be executable on the diversity of platforms which may be encountered. Providing an executable application for each platform is called porting the application. The porting problem is well known in the development of applications used in local environments and can be dealt with, albeit with considerable effort. Once an implementation language has been chosen, a compiler of that language is required for each of the possible platforms. The reference implementation is then adapted to comply with the technical particulars of a specific platform and compiled, yielding an executable for that platform. Ported applications are tightly integrated with the operating environment, and its user interface has a "look and feel" associated with the platform. This is considered a drawback, since users may be exposed to different platforms, and will be troubled by such differences.

Java application: The Java [Java] language offers an attractive alternative to porting as described above, since Java applications are not compiled for a specific platform. The compiled format of Java is platform-independent bytecodes (in the `class`-file), which are interpreted by a standardized Java Virtual Machine [Lindholm]. The JVM is platform dependent, but implementations thereof are available for all the popular platforms. The Java Runtime Environment (JRE) contains the JVM and the standard language `class` files, and is available free of charge. After installing the Java Runtime Environment on a particular workstation, the `class` files of the distributed analysis model can be copied to the workstation for execution. Since the `class` files are small, this step is not problematic. Java also addresses the look and feel problem by supporting standardized types of look and feel. The pilot implementation was done in Java.

7.2 Activation of references

The core issue which makes collaboration possible is that parameters can be introduced into a model and connected to other parameters in an uncomplicated way. The technique of using the external (persistent) identifiers of parameters to define relations between objects, referred to in section 5.3.1, provides the solution to this problem. In the collaboratory, however, the parameters are versioned. The assimilation of a versioned parameter is described below.

Selection identifier: Within the scope of a project, each analysis parameter used in the distributed analysis model has a unique string identifier called its persistent identifier (pid)[1]. Furthermore, the analysis parameters are versioned[2]. A particular version of a parameter may be retrieved from the project analysis parameter database using its selection identifier (sid), which is a combination of its persistent identifier and its version number[3].

Application object map: A specific version of a parameter can be selected in the database, transported to the requesting workstation, and instantiated in the address space of the distributed analysis application executing on that workstation. However, the parameter is entered into the object map using only its persistent identifier as key, as shown in figure 7-1.

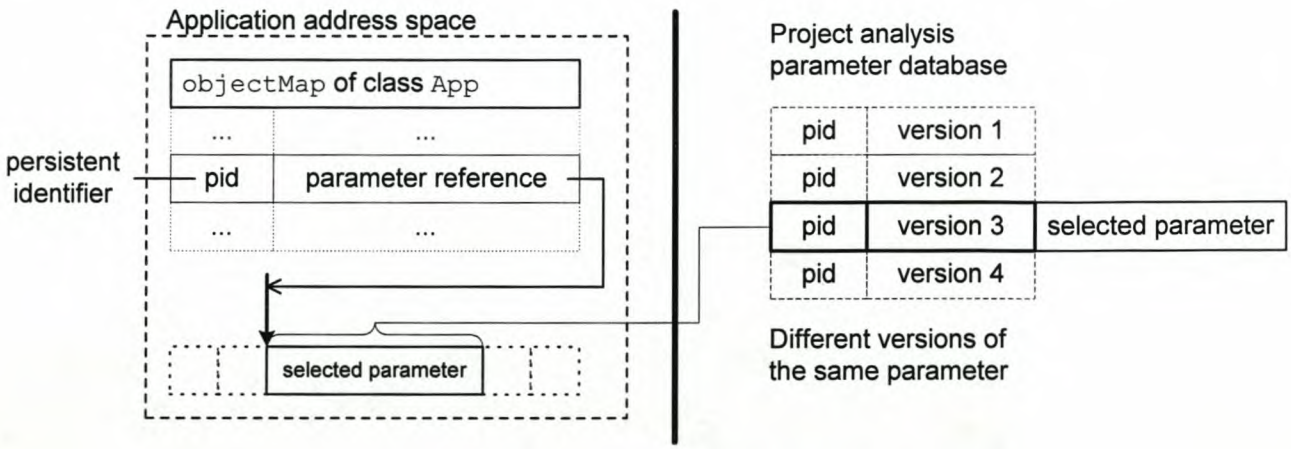


Figure 7-1: Object map in application address space

Loaded version: The way of activating references developed in section 4.3, based on using the persistent identifiers and the `objectMap` of class `App`, is left unchanged. For versioned parameters, this implies that the loaded version of the parameter is the one used for all operations involving the parameter.

[1] see section 6.1
[2] see section 6.2
[3] see section 6.2

7.3 Collaboratory services

A structural analysis collaboratory is formed to execute the analysis task of a project. The services of the collaboratory provide the essential functionalities which make collaborative work possible. The project setting, and the functions of the services within this setting, are explored below.

7.3.1 Project setting

Registration: Collaboratory services are provided within the context of a project, and can only be used by members of the analysis group that are responsible for the structural analyses of the specific project. This implies that some form of access control is required. Logging in with a username and password is a practical and standard form of access control used in computer environments, and is suggested for the analysis collaboratory.

Project attributes: Certain attributes of the project play an important role in the management of the analysis parameters, for example in the assigning of persistent identifiers[1], and in control of access to individual parameters[2]. The project setting is characterised by the following attributes:

- Project identifier: The project identifier is a string of characters and numbers which uniquely identify the project. The use of project identifiers is standard practise in design offices.
- Members: Members of the analysis group. The members are identified by their respective usernames.
- Tasks: The necessary structural analyses comprise of a number of analysis tasks. Tasks are planned as part of the analysis schedule, which is part of the project schedule.
- Task groups: In a collaboratory, task groups are assigned to execute tasks. Each task group comprises of a number of members that collaborate in the execution of a specific task.

[1] see section 6.1
[2] see section 6.3

7.3.2 Project analysis parameter database service (PAPD service)

The PAPD service provides members of the analysis group with the possibility to check project analysis parameters into the project analysis parameter database (PAPD), and to check parameters out of the database for use in models being processed at the workplaces. The PAPD is the common information space of the collaboratory, and access to this space is provided and controlled by the PAPD service.

Configuration: The project analysis parameter database is a monolithic database residing on the project server computer[1]. The PAPD service consists of two parts, namely the PAPD server and the PAPD clients. Details of the database model are hidden from the users at the workplaces by the PAPD server which communicates with the database on the one side, and the PAPD clients at the workplaces, as shown in figure 7-2. The distributed programming model which is used to implement the service determines the details of the transfer of data between the PAPD clients and server. All actions performed by the service are initiated on the client (workplace) side.

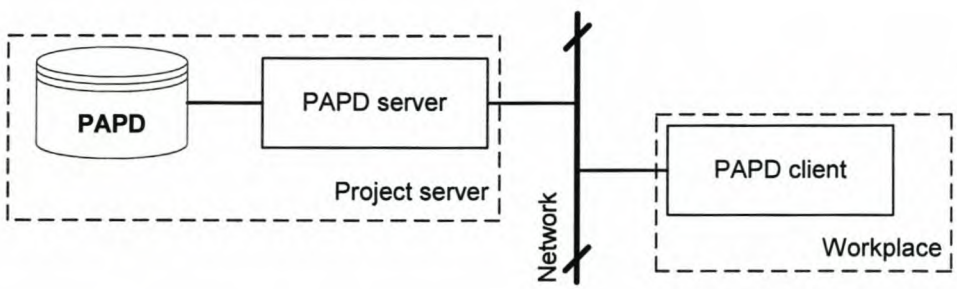


Figure 7-2: Project analysis parameter database service

7.3.2.1 Checking parameters into the project analysis parameter database

What is meant by checking a parameter into the project analysis parameter database (PAPD) depends on the category and state of the parameter. Parameters in use at the workplaces may belong to any of the following categories:

- **New parameters:** New parameters are parameters that were created at a workplace, and have never been part of the project analysis parameter database. Checking new parameters into the PAPD entails transporting data describing the state of the parameter to the project server, and storing it in the PAPD.
- **Consistent use parameters:** These are parameters that have been checked out of the PAPD for use in consistent mode (see section 7.3.2.2 below). If the state of the parameter has been changed, a new version of the parameter has to be created, and this new

[1] see section 6.5.2.3

version's state is transported and is stored in the PAPD. If the state of the parameter at check-in time is the same as its state at check-out time, i.e. the version of the parameter has not changed, no data needs to be transported since it is already present in the database. However, the PAPD service is notified that use of the parameter is terminated. This notification is passed on to the consistency service[1].

- **Free use parameters:** These are parameters that have been checked out of the PAPD for use in free mode. If the state of the parameter has been changed, the same conditions as for the changed consistent parameter applies (see directly above). If the parameter's state has not changed, no action is required.

Transfer of new parameters, and new versions of existing parameters to the PAPD requires transport of data over the network from the workplace to the project server. Clarity about what data needs to be transported is essential.

Fundamental attributes: The analysis parameters of the project are instances of the following classes[2]:

`Model, Equation, Node, Local, Support, Material, Element, NodeLoad, LineLoad, AreaLoad.`

The PAPD must have the capacity to capture the state of such instances. Certain attributes of objects have a transient state, i.e. they hold information for a certain duration of time, but these values need not be reflected in the persistent state of the object. In order to restrict the database storage size, but especially to minimize the amount of data that need to be transported over the network, it is important to store only the fundamental attributes of the analysis parameters. In section 6.2 the issue of controlled versioning is discussed, and the concept of fundamental change was introduced. Table 6-1 lists the attributes of each parameter class which might incur a fundamental change. Table 7-1 below lists the same attributes as the ones for which the persistence service has to make provision. It is important to assure that the mechanisms which are responsible for writing and reading object attributes in the network transfer data stream ignore the transient attributes. In Java, for example, this can be accomplished by implementing the interface `java.io.Externalizeable`.

[1] see section 7.3.4
[2] see section 5.2.1

Parameter	Fundamental attributes for storage
`Model`	Selection identifiers of all component parameters. (See section 5.3)
`Equation`	Primal and dual vectors (Cholesky factor on request).
`Node`	Coordinates, system indices, local system persistent identifier
`Local`	Rotation angle or direction cosines
`Material`	Young's modulus, Poisson's ratio
`Element`	Persistent identifiers of nodes and material, thickness
`NodeLoad`	Persistent identifier of node of application, load intensity
`LineLoad`	Persistent identifiers of line endpoint-nodes, load intensity
`AreaLoad`	Persistent identifier of element on which applied, load intensity
`Support`	Persistent identifier of node where applied, support conditions

Table 7-1: Fundamental attributes of analysis parameters

Access-control data: A project analysis parameter is instantiated and used within a certain context. The context determines which default access control rules apply to the parameter when it is stored in the project database. The context is known by the PAPD service since members of the analysis group have to register with the project's collaboratory services. For storage of parameters, the project attributes below are important:

- A parameter is used by a specific member of the analysis group, identified by his project `userName`. The member belongs to a certain `taskGroup` within the analysis group of the project.
- A parameter is used in the execution of a specific `task`, which is the responsibility of a `taskGroup`.
- Certain default access control rules apply to the parameter, which may be overridden by the owner, identified by his `ownerId`.

Check-in summary: In order to check a parameter into the PAPD, access to the PAPD service must be obtained by logging in to the project for which that service is provided. This allows verified members of the analysis group to establish communication with the PAPD server. Amongst the login details are the member's `userName`, and the `task` that he is working on. It can be verified that he is a member of the `taskGroup` responsible for the given task. If he instantiates new parameters, or creates new versions of existing parameters, he is the owner of the new parameters or versions of parameters. Consequently his username is assigned as the `ownerId` of these parameters. The procedure for checking in a parameter, and the data involved in the process is summarized below. Since all parameters are subclasses of class `AppObject`, they can be treated in an equivalent way.

1. A parameter, which exists in the application running on the client workstation, is identified to be checked into the PAPD.
2. All parameters are aware of undergoing fundamental change through the implementation of an interface[1]. The parameter is queried regarding modification.
3. If the parameter has not been modified fundamentally:
   a) If it has been checked out for use in free mode, no further action is taken.
   b) If it has been checked out for use in consistent mode, notification is sent to the PAPD service that use of the parameter is terminated.
4. If the parameter has undergone fundamental change, or it is a new parameter, a new version of the parameter has to be created before it can be checked into the PAPD. Apart from its fundamental attributes, the following data is stored with each parameter:
   - For the purpose of access control, its `ownerId` and the identifier of the `taskGroup` within which it was created, is required[2]. These particulars are known to the PAPD service as described above.
   - The `accessCode` of the parameter[2] is determined from the default configuration for the `taskGroup`, but can be overridden by the owner. In the latter case the user-specified `accessCode` has to be sent together with the parameter.

7.3.2.2 Checking a parameter out of the project analysis parameter database

Checking out a parameter is the action whereby a member of the analysis group obtains the state of an existing parameter from the project analysis parameter database and creates an instance of that parameter in the application executing on his workstation. As such, it is the opposite of the checking in action described above.

Access control: Two restrictions apply when a parameter is checked out, namely control of access to the the collaboratory services, and control of access to the parameter itself. Access to the collaboratory services is obtained by logging in as described in the project setting above. Access control to individual parameters is described in section 6.3.

Check-out mode: The access control settings of a parameter differ depending on the mode, i.e. either free mode or consistent mode, in which the requesting client wants to use the parameter. Consequently the check-out mode has to be specified when a parameter is requested.

Check-out summary: Members of the analysis group will check parameters out of the project database in order to help them perform their respective tasks more productively. The navigation problem, i.e. the problem of finding useful parameters, is discussed in chapter 8.

[1] see section 6.2
[2] see section 6.3

Given that a parameter in the PAPD has been identified for use, and the mode in which it is to be used has been decided on, the procedure for retrieving it, and the data involved in the process, are summarized below:

1. Using the parameter's selection identifer as key, the PAPD service retrieves the parameter's data from the database. The access control tag of the parameter is part of that data, and can be analysed to check the access rights of the requesting client. The `userName` and `taskGroup` of the requesting member is required to check his access rights, and these are known since he had to log in to work on the project. If the requesting member does not have the required access rights, he is informed and no further action is taken.
2. If the parameter has been requested for use in consistent mode, the PAPD service will notify the central consistency service[1] and provide the parameter's selection identifier and the username of the member who will be using it.
3. The data necessary to describe the state of the parameter is transferred to the client workstation, where a component object of the appropriate class is instantiated and its attributes are set to the data received. The particulars of the distributed programming model determine the details of the transfer of the parameter.
4. In the distributed analysis application running on the client workstation, the newly instantiated object is entered into the `objectMap` of class `App` using its persistent identifer as key. If an object with the same persistent identifier is present in the application, it is replaced by the new object.

[1] see section 7.3.4.2

7.3.3 Persistent identifier service

The persistent identifier service (PID service) issues the persistent identifiers of all analysis parameters of a project. Its principal purpose is to ensure that all identifiers are unique within the scope of the project.

Configuration: A PID service could be configured in two ways:

- Centralized service: In this case the PID service comprises of a PID server, installed on the project server computer, and PID clients installed on the workstations at the workplaces. The PID server generates identifiers and distributes them to the PID clients.
- Decentralized service: In this case the complete PID service is installed on each of the individual workstations at the workplaces.

The two configurations, shown in figure 7-3, are discussed below.

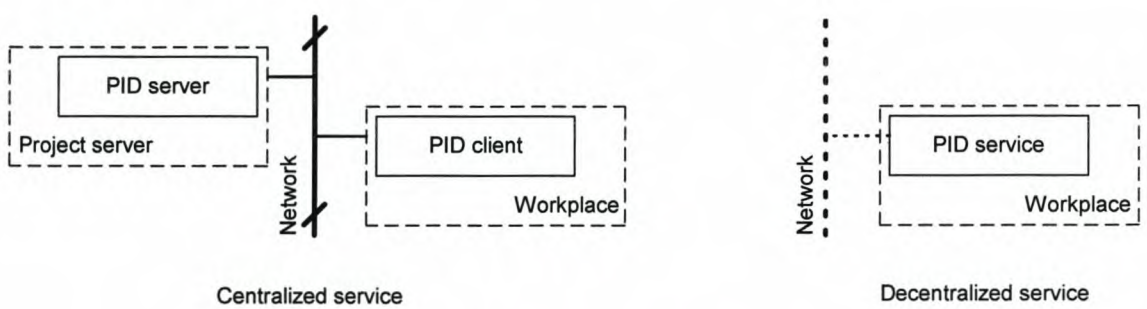


Figure 7-3: Persistent identifier service

Decentralized PID service: The decentralized PID service has the advantage that it can issue identifiers in the same runtime environment as the requesting application, consequently identifiers can be made available without the latency associated with network communication. Uniqueness of identifiers issued on a workstation can be guaranteed by using, for example, the physical address of the workstation's network card or it's internet protocol (IP) number. However, this has the disadvantage that identifiers become longer, without adding any useful meaning to the identifier[1]. To avoid this, allocated identifiers have to be registered with a central databank which has to be accessed to assure uniqueness before new identifiers are issued. Such an alternative offers no advantage over the use of a centralized PID service.

Centralized PID service: A centralized PID service with the PID server installed on the project server and PID clients at the workplaces is easier to configure, administer and maintain than a decentralized service. However, there is the risk of communication breakdown between the server its clients due to network or server failure. Another disadvantage of the centralized service is the latency associated with the transfer of identifiers over the network.

[1] see section 6.1

Since identifiers are required during the modeling phase in interactive worksessions, this issue has to be addressed. Both of these disadvantages can be overcome in two ways:

- **Buffering:** The PID client can be configured to maintain a cache of identifiers in a local buffer, e.g. to hold in store 1000 Node and Element identifiers, 100 Support identifiers, etc., for use at the appropriate time. At the time of instatiation of new parameters, these identifiers would be available without having to rely on a network connection at that time, and without latency due to network communication.
- **Temporary identifiers:** Modeling methods can be extended to support the use of temporary identifiers, with the possibility to map the temporary identifiers to persistent identifiers when the latter become available.

The disadvantages of a centralized PID service can be overcome relatively easily, especially by buffering. Its advantages make the centralized persistent identifier service the implementation of choice.

7.3.4 Consistency service

The consistency of project analysis parameters is discussed in section 6.4. The consistency service has to provide the functionality described there. In summary, it entails the gathering and distribution of modifications to consistent parameters amongst the members of the analysis group that are using these parameters.

Configuration: The consistency service of the collaboratory requires a central service which is aware of all parameters that are used in consistent mode, and who are using them, and has the capacity to receive changes to parameters from any of the workplaces and notify other users about these changes. At the workplaces a consistency service is required which forwards changes made to consistent parameters at the workplace to the central service, and which can receive and process changes received from the central service. The service described implies a peer to peer configuration as shown in figure 7-4 below.

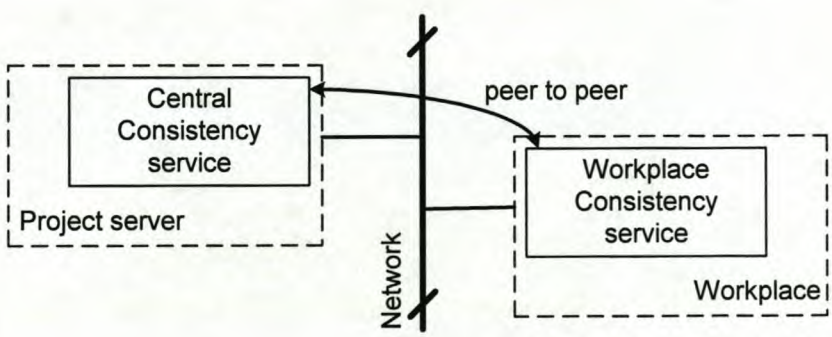


Figure 7-4: Workplace and central consistency services

7.3.4.1 Workplace consistency service

Tasks: The workplace consistency service performs the following two tasks:

- Forward changes to consistent parameters from the workplace to the central consistency service. These changes are made by the member of the analysis group who checked out the parameter in consistent mode.
- Receive and process changes arriving from the central consistency service. These changes are made by collaborating members, working at other workplaces, to consistent parameters which are used at those workplaces as well as at the workplace under consideration.

Demands on parameters: In order to perform its tasks, the workplace consistency service places the following demands on parameters which are used in consistent mode:

- **Consistent use awareness:** Parameters have to register the fact that they are used in consistent mode, e.g. through a boolean attribute which is true when the parameter is used in consistent mode, and false otherwise. At the time the parameter is checked out of the PAPD, the mode of usage has to be specified, and can be recorded when the parameter is instantiated in the workplace application.
- **Change event generation:** Parameters have to recognize and report fundamental changes to themselves. The issue of registering fundamental changes has been raised in connection with the versioning of parameters as well[1], and the fundamental attributes have been pointed out. Changes to attributes are made through appropriate `set`-methods, and these methods can be extended to recognize fundamental changes, register the fact that such a change has taken place, and forward the method name and the old and new parameters as a change event to the consistency service for further processing.
- **Update capability:** Parameters have to be able to update attributes in reaction to changes which occurred at another workplace. An update is a fundamental change to a parameter brought about using a change event for the parameter received via the consistency service. The event was generated due to the same change taking place at another workplace on a parameter with the same selection identifier. In order to prevent a feedback loop, parameters should not report updates to the consistency service.

Listener pattern: Parameters have to report fundamental changes to the workplace consistency service for processing, as described above. Reporting such change events is best achieved through using the listener pattern [Gamma 1995]: Support for generating and forwarding fundamental change events should be provided to each parameter that is used in consistent mode. The support generates a change event when a fundamental change takes place on the parameter it supports. It then forwards the change event to all listeners that

[1] see section 6.2

have registered with it to receive those events. In this case typically only one listener, namely the workplace consistency service would register to receive the change events.

Buffering local changes: The workplace consistency service is responsible for forwarding local change events to the central consistency service. The forwarding requires an operational network connection and a running central consistency service, both of which cannot be garuanteed at all times. Until an active connection to the central consistency service can be made, change events have to be buffered in a list which is traversed as soon as the connection has been established.

Buffering received changes: The workplace consistency service receives change events from other workplaces where the same parameters are used in consistent mode. The consistency service could be configured to automatically update a parameter as soon as a change event for it is received. However, it is considered advantageous to offer the local user the possibility of effecting the update at a later point in time, or of rejecting the change alltogether[1]. Incoming change events should be buffered in a list which the user can work through at an appropriate time.

7.3.4.2 Central consistency service

Tasks: The central consistency service performs two main tasks:

- Receive all fundamental change events pertaining to consistent parameters from the workplace consistency services.
- Forward each fundamental change event to all the workplace consistency services where the affected parameter is used in consistent mode, except to the source of the change event.

Information: In order to perform these tasks, the central consistency service has to maintain the following information about each analysis parameter which is checked out for use in consistent mode:

- The parameter's selection identifier. For each parameter checked out for use in consistent mode, the PAPD service forwards to the central consistency service the selection identifier of the parameter, and the username of the member of the analysis group to whom the parameter is provided[2].
- A set of usernames of members of the analysis group that have checked out the parameter for consistent mode usage. The usernames are known, as described directly above.
- A list of all changes to the parameter. This list is the union of all change events received from the workplace consistency services, whose task it is to forward all changes made at the workplaces to the central consistency service.

---

[1] see section 6.4
[2] see section 7.3.2.2

- A record of which changes have been reported to each user. Each user must be notified about all changes of which he has not been the source.

Consistency support termination: A parameter is considered to be used in consistent mode from the time it is checked-out in consistent mode by the first user, until the time it is no longer used in consistent mode by any member of the analysis group. A user may terminate his use of a consistent parameter in two ways:

- Check the unchanged parameter in using the functionality of the PAPD service provided for that purpose[1]. The PAPD service should instruct the central consistency service to remove the user from the user list of the parameter. It is important to note that the parameter is not removed from the user's working environment by this action, but his continued usage of the parameter is in free mode.
- Change the parameter fundamentally, and then create a new version of the parameter. Since the selection identifier of the new version differs from that of the original parameter, the parameter should, as part of the process of creating its new version, notify the central consistency service to remove the user from the user list of the original parameter.

[1] see section 7.3.2.1

7.3.5 Versioning service

The project analysis parameters are versioned. User controlled versioning, and the `version` attribute of parameters are described in section 6.2. The task of the versioning service is to support the creation of a new version of a parameter by providing the functionality to update the parameter's version attribute at the time the new version is created.

Version attributes: A version, as described in section 6.2, has three attributes, namely an integer `number`, a list of integer numbers representing its `history`, and a long (8 byte integer) `timestamp` reflecting the time of versioning.

Configuration: A central versioning server for the collaboratory is necessary since more than one member of the analysis group may check out the same parameter, change it fundamentally and create new versions of it. The versioning server should be installed on the project server computer, and be accessible to versioning clients running on workstations at the workplaces, as shown in figure 7-5. Details of the distributed programming model determine how the transfer of data between the clients and the versioning server takes place.

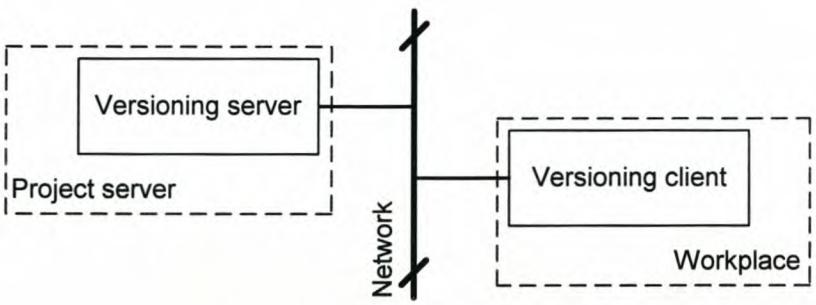


Figure 7-5: Versioning service

Versioning client: The task of the versioning client is to update the version attribute of an analysis parameter at the parameter's request. The parameter's persistent identifier, and the existing version are required to perform the update. With the persistent identifier as input, the versioning client can retrieve a new version number and timestamp from the versioning server. The update of the version attribute then entails adding the existing version number to its history list, and changing the version number and timestamp to the values that were retrieved from the server.

Versioning server: The task of the versioning server is to provide the next available version number for a given parameter, and the time at which the number is issued. To perform this task the server has to register the last version number issued to each parameter. A hash map, mapping the persistent identifier of a parameter to the last number issued to it, pro-

vides a fast way of retrieving the last version number of a parameter. The version time is provided by the project server's clock.

Communication breakdown: Temporary loss of communication between a versioning client and the versioning server is not considered critical since all normal analysis activities at the workplace can continue in spite of such a breakdown. The only activity which cannot take place before the necessary new versions of parameters have been created is that of checking the parameters into the project analysis parameter database. As soon as the versioning service is functional, the updating of parameters' version attributes can be completed, after which the parameters can be checked into the PAPD.

8 User interface

The scope of a distributed analysis model, and the distribution of its parameters and methods have been discussed in the previous chapters. The focus there was on providing the data-level and functional-level infrastructure to support structural analysis in a distributed collaboratory. In this chapter the focus is on supporting collaborative analysis at the process level, i.e. supporting members of the analysis group in the collaborative creation, processing and evaluation of analysis models through the provision of a coherent user interface to those models. This interface has to support users in dealing effectively with two groups of functionality, namely

- the technical aspects of finite element structural analysis, and
- collaboration with other members in the execution of analysis tasks.

Collaboration awareness: User interfaces have traditionally been designed to support a single user working in a local environment. Interaction with other users was limited to the use of mechanisms available at the operating system level, e.g. the exchange of files using the file transfer protocol (ftp). The adaption of a single-user application for use in a multi-user environment is often made by enabling users to share "collaboration transparent screens" [Bentley 1994]. For the structural analysis collaboratory, however, a solution is required which allows members of the analysis group to use and view analysis parameters in independent ways, while maintaining collaboration awareness in the sense that the distributed work effort is reflected on the user interface. This can be achieved by exploiting the potential offered by the distribution of parameters and methods described in the previous chapters

Main features: No attempt is made to list all options and actions that a user interface could or should have. Instead, the main features of a collaboration-aware interface for the distributed analysis application are discussed. It is argued that the activities and actions supported by typical single user interfaces could be extended to provide those features.

8.1 Support of user programming

Fundamental problem: A central issue in the design of a user interface is the possible disparity between activities and options that were foreseen by the software developer and provided for in the interface, and those which the users in fact want to perform and require. The activities and options comprising the collaborative structural analysis process are copious, and cannot be provided for by enumeration. Consequently, functionality provided at the user interface can only make provision for standard cases. More general cases can either not be dealt with, or have to be approximated using an available abstract construct. The degree of abstraction which is supported at the user interface level determines the generality of user options. The user's options could be increased significantly by providing, from the interface, access to objects of the application through the use of a programming language.

Restrictions: Standard user interaction with a finite element model and its component objects is indirect, i.e. via a user interface, and, as a rule, the user is not provided programmatic access to the model and its components. This restriction is due to limitations imposed by both the implementation language and the internal structure of the application. If, for example, the programming language does not allow the loading of a class upon demand, options with regard to user programming are limited. Furthermore, the internal structure of the application may be complex, and the model and its components difficult to access.

Model: The finite element model described in chapter 4 is simply a set of sets of component objects, and it is easy to reach the component objects and interact with them. If, for example, nodes with certain coordinate values are to be found, the `nodeSet` of the model can be traversed. For each node its coordinates can be retrieved using the `getCoordinates()`-method of interface `IcNode` (which is implemented by class `Node`), and compared with the given values. Furthermore, the structure of the application reduces the complexity of establishing relations between objects to the level of doing it in terms of external persistent identifiers, and existing objects can be reached via their persistent identifiers. As a result a user can, in principle, interact with the model at the source code level without too much difficulty.

UserOperation: The Java programming language has the capacity to load a class upon demand. Consequently, a user programmed class can be loaded by an instruction from the user interface at any point in time. If this class implements, for example, a Java interface called `UserOperation`, which specifies the method `execute(Model model)`, an instance of the class can be created, cast to the `UserOperation` datatype and its `execute`-method called. In this way any number of user-programmed actions can be performed from the user interface, the only restriction being that the user-classes must have unique names and implement the interface `UserOperation`.

User programming: The combination of a programmer friendly application and model structure, and the stated features of the Java language, creates an opportunity for extending the

user's options significantly through source code-level interaction with the model. Opportunities where user programming could be useful are pointed out in the following sections.

8.2 Explicit collaboration

In the execution of their respective tasks, members of the analysis group perform various actions which are directly aimed at collaboration, e.g. checking parameters into and out of the project analysis parameter database. These are explicit collaborative actions. User interface support of explicit collaboration is discussed in this section.

In section 8.3, user actions which focus on the technicalities of finite element analysis are discussed. Some of these actions, however, have an impact on the collaboratory, and implicit collaboration takes place during their execution.

8.2.1 Project analysis tasks

The analysis collaboratory is formed to perform the structural analysis tasks of a building project[1]. Members of the analysis group are assigned to taskgroups, with each taskgroup responsible for given analysis tasks. Consequently, all activities that members perform during a worksession are aimed at the execution of a particular task of a particular project.

The user interface has to allow a user to:

- **Choose a project on which to work:** This is necessary since a user may be involved in more than one project.
- **Login to the chosen project:** Unauthorized access to a project's analysis information has to be prevented. Legitimate members of a project's analysis group each have a unique username and password which allows access to the collaboratory services of that project
- **Choose a particular task to work on:** Being a member of an analysis group, the user is responsible for given tasks of the project. Choosing a particular task to work on places him in the context of the taskgroup (of which he has to be a member) that is responsible for the task. This is important, since control of access to certain parameters may be different for members of a taskgroup than for general members of the analysis group.

8.2.2 Project analysis parameters

Mutual exchange of project analysis parameters form the core of the collaboratory. Exchange of parameters take place via the project analysis parameter database (PAPD). Access to the PAPD is provided by the PAPD service[2]. Through a functional PAPD service, a logged in member of the analysis group can check project analysis parameters into and out of the project analysis parameter database.

---

[1] see section 5.1
[2] see section 7.3.2

In support of the above, the user interface has to provide for the following actions and options:

- **Activate and terminate the PAPD service:** A member of the analysis group may work for long periods of time without any need for interaction with the PAPD. When required, such interaction demands explicit action by the member, which should start and end by activating and terminating the PAPD service.
- **Check parameters into the PAPD[1]:** Through this action a member makes new parameters, or new versions of existing parameters, available for use by other team members. He has to exercise options pertaining to the following:
  - ***Selection of local parameters:*** The complete set of project analysis parameters in the user's runtime are possible candidates to be checked into the PAPD, and all of these are uniquely identified in the runtime by their persistent identifiers. Selection options would be:
    - *Single parameter:* Identified by its persistent identifier.
    - *Component set of the model:* All parameters belonging to a set identified by the name of its component class, e.g. the Node set, the Element set, etc.
    - *Complete model:* All parameters comprising the model.
    - *User programming:* Execute a user-programmed operation which selects parameters according to some scheme, places them in a set and submit the set as parameters to be checked in.
  - ***Setting access codes:*** If an access code is not specified, the default access code will be assigned[1]. However, the user requires an option to set a different access code, which remains in place until it is changed in turn, or set back to the default.
- **Check parameters out of the PAPD[2]:** Through this action a member retrieves existing parameters from the PAPD in order to use them in some way in a model he is working on. He has to exercise options pertaining to the following:
  - ***Usage mode:*** Parameters are checked out of the PAPD for use in consistent mode or free mode[3], which must be specified by the user.
  - ***Selection of PAPD parameters (Navigation):*** The complete set of parameters in the project analysis parameter database are possible candidates for selection, and all of these are uniquely identified in the database by their selection identifiers[4]. The problem facing members of the analysis group is to identify parameters that would be useful for a particular purpose, i.e. they have a navigation problem. Since the usefulness of a parameter depends on the user's particular purpose for it, navigation aids can never be comprehensive. Support for browsing the PAPD is helpful in general, since it

[1] see section 7.3.2.1
[2] see section 7.3.2.2
[3] see section 5.2.2
[4] see section 6.2

helps members identify parameters for selection and strengthens the feeling of collaboration. The options below are considered useful in aiding parameter selection for a broad class of purposes:

- *Model:* In finite element analysis work, the finite element model is the fundamental unit of information — useful parameters are components of useful models. Within a model, helpful selection options are the following:
  - Component set: The component classes are assigned to sets, with the result that a specific set of components parameters of a model can easily be identified and selected, e.g. the nodes of a given model.
  - Union of sets: Through the union of component sets useful supersets can be formed, e.g. the loading set as the union of the nodeload, lineload and areaload sets.
  - Condition set: A set of parameters complying with a certain condition, e.g. all elements of a given material, or the elements that are directly supported, or the nodes where loads are applied.
  - Relation: A set of parameter pairs that are tied by a given relation, e.g. elements that are directly connected to each other, nodes and their corresponding node loads, etc.
- *Dependency:* A single parameter is simply selected by specifying its selection identifier, i.e. its persistent identifier and version number. Useful extentions to the selection of a single parameter would be to select a parameter plus all parameters that depend on it (excluding the model it belongs to), or all parameters that it depends on.
- *Origin:* A parameter is created to be used in a particular task, and it has an owner. These attributes may be used to create useful selection sets, e.g. all nodes created for a particular task, or all parameters created by a particular member of the analysis group.
- *Geometry:* Structures are to a large extent characterized by regular geometry, and engineers tend to think in terms of geometry. Supporting selection of parameters in terms of geometric properties is essential, e.g. all nodes and elements that fall inside a given bounding box.
- *User query:* A relational database with support for the Structured Query Language (SQL) has been proposed for the project analysis parameter database[1]. For retrieval of information from a relational database, SQL has the powerful SELECT-FROM-WHERE statement. If the user has sufficient information about the database format, he could construct SQL queries for very purposeful selection of parameters. The following SQL query will, for example, retrieve the persistent identi-

[1] see section 6.5.2.3

fiers and version numbers of all Elements which are connected to a Node with identifier N1:

```
SELECT pid, versionNumber FROM tElements
WHERE ( (nodeId1='N1') OR (nodeId2='N1') OR (nodeId3='N1') );
```

8.2.3 Dynamic profile

The members of a structural analysis collaboratory need to be informed about each other's activities pertaining to the analysis tasks of the project that they are working on. This encourages collaboration and enhances the productivity of the group as a whole. Although much of the communication required to distribute this information could take place over standard media like telephone and email, it should be supported at the user interface level as well, for example in the form of a member bulletin board and a chat service.

Member bulletin board: The bulletin board could contain messages placed by members of the analysis group and the analysis management team. It would have sections for the different task groups where notes of mutual interest are placed. Important notices would be those indicating that parameters have been checked into the project database. To assist potential users of these parameters, notices should supply navigation aids like the selection identifier of the model containing the parameters, and a short description indicating special characteristics of the parameters.

Chat service: The chat service would give a snapshot of activity within the collaboratory by listing the last login time, logout time and task worked on by each member. It is useful in situations where members might want to exchange ideas without the delay associated with email communication. Two members should be able to keep communication between them exclusive, or allow or invite more members to join in the formation of a chat group.

8.3 Finite element analysis

Implicit collaboration: The description and execution of an analysis task in a local, single-user environment was dealt with in chapter 4. A specific aim in the design of the structural analysis collaboratory was that members of the analysis group could continue work in spite of a temporary loss of network communication, i.e. in a situation similar to the local, single-user environment. As a result the distributed analysis application, and specifically its user interface, has to encompass the functionality required in local environments, supplemented by extentions which deal with distributed collaboration. Support for explicit collaboration was discussed in the previous section. In this section the focus is on implicit collaborative actions. These actions are aimed primarily at parameters in the user's local environment, but they nonetheless have an impact on the collaboratory as a whole.

8.3.1 Description of analysis parameters

The descriptive part of the analysis task may be influenced significantly by the use of parameters that were created by team members. Whether such parameters are used or not, (supplementary) actions aimed at describing the analysis model are performed by the user in a local context. These actions entail the creation of new parameters and the editing of existing parameters. The parameters involved are described in section 5.2.2. Two standard user interfaces for creating parameters and editing their content are briefly described. The collaboration awareness of the user interface with respect to creation and editing of parameters is then discussed.

Text input file: The use of a formatted text input file to describe an analysis task was the natural substitute of the original punch card systems, and stems from the time when computer graphics capabilities were very limited. Data is written in the input file using an application specific markup language, which is interpreted by the application and converted to its internal format for processing. The use of an input file has the advantage that it is an automatic form of persistence, or at least repeatability, of an analysis. Furthermore it is an unambiguous description that can be incrementally corrected and improved when results indicate that mistakes have been made. However, the user's options are always limited to what can be achieved with a text convention, and simple errors, like typing mistakes, may be difficult to uncover.

Graphics-based modeling tools: Current computer graphics capabilities have enabled developers to create powerful graphics-based modeling tools. Typically, these tools have the capability to generate parametric descriptions of geometric objects like lines, surfaces and volumes from a variety of descriptive formats. Input data is specified in either graphical or tabular form. The geometric objects have methods for generating a mesh of nodes and elements inside themselves and on their surfaces, for applying static and kinematic boundary conditions on their surfaces and along their edges, for specifying material properties, etc.

As such they can be used to create and edit finite element parameters for a very broad class of problems encountered in structural analysis. Graphics-based modeling tools represent the current state of the art in finite element modeling, are widely used, and would be part of a commercial distributed analysis application.

User programming: The use of a text input file limits the user's options regarding the creation and editing of objects to the structures allowed for in the markup language, as stated above. Graphics-based modeling tools are less restrictive, but cannot provide complete generality of options. Extending the user's options by supporting his use of a programming language, as described in section 8.1, would represent a valuable extention with respect to the creation and editing of parameters.

Collaboration awareness: During the modeling phase of creating and editing analysis parameters, implicit collaboration takes place with respect to the following activities:

- **Assigning persistent identifiers to new parameters:** When new parameters are created, their persistent identifiers have to be issued by the collaboratory's persistent identifier service described in section 7.3.3. This service allows for cacheing of identifiers, or the use of temporary identifiers, to overcome the problem of communication breakdown when identifiers are required in an interactive worksession. User interaction is required in both cases, which implies consideration at the user interface level. If cacheing is used, the user should be able to configure the cache sizes for the different component classes, and be able to query the state of the cache at any time. If temporary identifiers are used, the user should be able to query the state of the PID service and issue an instruction to substitute temporary identifiers with persistent identifiers when the PID service is available.
- **Creating new versions of parameters:** User controlled versioning, described in section 6.2, requires explicit action by the user, which has to be provided for by the user interface. The user should be able to create new versions of single parameters, sets of parameters, or of the model as a whole. Since the versioning service of the collaboratory[1] is responsible for updating version attributes, the user has to be able to control the status of this service.
- **Changing consistent parameters:** Changes to fundamental attributes of a parameter used in consistent mode are propagated by the consistency service[2]. Forwarding of local changes by a user takes place without user interaction, except start/stop control over the consistency service. However, changes reported from other locations have to be considered by the user. Project policy may dictate that all reported changes be implemented, or may allow user discretion regarding accepting or rejecting a reported change. In any event the user has to receive clear indication about the parameter involved and the na-

[1] see section 7.3.5
[2] see section 7.3.4

ture of the change. Reporting has to be done in an unobtrusive manner which allows the user to deal with reported changes at a convenient time.

8.3.2 Execution of the analysis algorithms

The algorithmic phase of an analysis entails the assembly and solution of the system equations, followed by the transfer of results to the nodes and supports, as described in sectoin 4.5.2. This process may take place on the member's local workstation, or on a server computer, as described in section 5.3.3.2.

Local execution: Standard execution of an analysis step on a local workstation requires no explicit collaborative action by the user, and implicit collaboration is limited to obtaining an identifier for the `Equation` that is created during the process. The `perform()` step of the analysis described in section 4.5.2 executes a number of procedures required for a complete analysis. These are:

- `setReferences()`: Connect the component objects of the application by reference.
- `computeSystemMatrix()`: Assemble the system (stiffness) matrix.
- `computeSystemVector()`: Assemble the system (load) vector.
- `setStatusVector()`: Create the equation's status vector.
- `decompose()`: Perform a Cholesky decomposition of the system matrix.
- `solve()`: Compute the primal and dual solution vectors of the system.
- `saveResult()`: Transfer solution to the nodes and supports of the model.

The user interface, however, should allow for execution of each of these procedures individually. This would give the user more options, and is in fact necessary when a model and its solution vectors are retrieved for interpretation of results, as described below.

Server execution: The scenario for using a particular server to perform the analysis computations is depicted in section 5.3.3.2, and requires no special actions on the user interface.

8.3.3 Interpretation

Efficient interpretation of a finite element model's geometry, the influences it is subjected to, and its resulting behaviour requires a user interface which supports a rich variety of ways in which to view the component objects of the model. A user interface that supports a simple alpha-numeric view of the description and behaviour of a finite element model is described in section 4.2. In commercial applications the nominal functionality described there would be extended by providing for different ways of sorting, conditional output, printing of maxima and minima, etc. Furthermore, extensive support for graphical views on model components is a standard feature of current single-user applications, and would be essential for a distributed analysis application as well. The discussion below, however, is limited to the influence of distributed collaboration on interpretation-supporting features of the user interface.

Interpretation of models in the distributed collaboratory takes place in two distinct modes, namely a single user mode and a collaborative mode:

1. **Single user interpretation:** In this case a user views the model with his own aims and purposes in mind, and does not specifically seek or require active collaboration from team members. The model under investigation may be a new one that is under construction at a workplace, or it may be an existing model that has been retrieved from the project analysis parameter database (PAPD). The user may be viewing either the description of the model, or its behaviour.
   - ***Description:*** Viewing the description of a model under construction at the workplace, or one that has been retrieved from the PAPD, may require execution of the `setReferences()` procedure[1] in order to establish connections between the model's component parameters. An evaluation of the model's description may be followed by modification of parameters, and the creation of new parameters, leading to implicit collaboration as described in section 8.3.1.
   - ***Behaviour:*** Viewing the behaviour of a new model simply requires a successful analysis step beforehand. If the behaviour of an existing model residing in the PAPD is to be investigated, the complete model, including its primal and dual solution vectors has to be checked out. The `setReferences()` and `saveResult()` procedures[2] need to be executed in order to connect the component parameters and transfer the analysis results from the primal and dual solution vectors to the nodes and supports of the model. During interpretation of the model's behaviour, no fundamental changes are made to any of the parameters, and no implicit collaboration takes place.

   Supporting a single user in the interpretation of a model in his local environment requires the user interface features aimed at collaboration described in section 8.3.1, and provision for executing the `setReferences()` and `saveResult()` procedures individually.
2. **Collaborative interpretation:** Occasions may arise where team members wish to actively collaborate in the evaluation of a model's geometry, influences, and behaviour, in order to verify its compliance with given specifications. Collaborative interpretation may take place either asynchronously, or synchronously:
   - ***Asynchronous:*** This form of collaborative interpretation can be applied to models which reside in the project analysis parameter database (PAPD). The collaborating team members will check the model out of the PAPD, view and evaluate it in single-user mode on their respective workstations, and communicate their findings using the collaboratory bulletin board[3] or email. If all of the team members have checked the model out in consistent mode, any changes made will propagate to the other members via the collaboratory's consistency service. Since the interpretation of the model effectively takes place in single user mode at each of the workplaces, the user inter-

[1] see section 4.5.2
[2] see section 4.5.2
[3] see section 8.2.3

face requirements are identical to the case for single user interpretation described above. It is expected that asynchrounous collaborative interpretation of the description and behaviour of models will be the standard case for the distributed analysis collaboratory.

- ***Synchrounous:*** Synchronous collaborative interpretation is the action whereby team members simultaneously view and discuss the model under consideration from their distributed locations. The basic requirements for this form of collaboration is that participants know what view the others have on their screens, that they are able to point at objects, and that they can communicate effectively, preferrably by voice. These requirements are met by applications aimed at the support of network-based conferencing, like Microsoft's Netmeeting product [Netmeeting], or WebEx's conferencing service [WebEx].

Asynchronous collaborative interpretation does not require user interface features over and above those already identified. For synchronous collaborative interpretation an infrastructure which supports network-based conferencing can provide the required functionality without an impact on the user interface of the distributed analysis application.

9 Pilot application

The scope of a distributed analysis collaboratory has been described in the preceding chapters. The focus was on preferred ways of providing a software infrastructure that would be capable of supporting members of the analysis group in the execution of the set of tasks comprising distributed collaborative structural analysis. Core aspects of the implementation of a pilot application for distributed analysis are described in this chapter. The aim of the pilot application is to establish a capacity for executing examples of collaborative analysis tasks, and to perform a quantitative evaluation of the proposed solutions. Assessing the results of such an evaluation requires an understanding of certain implementation details. The aim of the overview presented below is to provide that understanding. The examples and evaluation are described in the next chapter.

9.1 Scope of the application

In chapter 4, an implementation of finite element theory of thin plates was described. The structure of an application, the components of the finite element model, the model's structure and functionality, the analysis algorithms, and the solution of the system equations were discussed. The application described in chapter 4 is aimed at supporting a single user working in a local environment in the execution of an analysis. The application described here extends the single-user application in order to support the execution of analysis tasks in a distributed collaboratory. Consequently two areas of functionality are required, namely provision for the technical aspects of finite element analysis and support for collaboration.

Platform: Similar to the single user application of chapter 4, implementation of the distributed analysis model is done using Java. This avoids the porting problem and discrepancies in the appearance of the user interface[1]. However, Java specific communication technology, namely Remote Method Invocation (RMI), was not used[2]. The Microsoft Access relational database management service [Roman 1999] was used for the project analysis parameter database[3].

Limitations: With respect to the technical aspects of finite element analysis, the limitations of the distributed analysis application are the same as those described in section 4.2 pertaining to the single-user application. The other area of functionality, namely support for collaboration, is provided for by implementations of the collaboratory services described in sections 7.3.2 through 7.3.5. Similar to the approach taken in the implementation of the analysis part, these services were implemented without attempting to provide an extensive range of options which might be useful in engineering practice. This would have required a huge programming effort, without contributing towards reaching the aim stated above. Instead, the

[1] see section 7.1
[2] see section 5.3.4
[3] see section 6.5.2.3

implementation was directed at providing basic functionalities while using simple architectures and readily available technologies.

9.2 Structure of a distributed application

9.2.1 Objects, classes and services of a distributed application

Distributed application: A distributed application is the software which is installed on a set of computers connected by a communication network for the treatment of a specific set of tasks. In this implementation plate analysis in a distributed collaboratory is treated as if it were an application. Similar to the single user application, a distributed application which allows for the different analysis types and structural components comprising the scope of structural analysis described in chapter 2 would require many person years of effort.

Objects: A finite element model is constructed in the local address space of a distributed application running on a workstation at one of the collaboratory workplaces. The model is an instance of class `Model`, and is composed of objects that are instances of the classes `Node`, `Element`, `Support`, etc. Some of these component objects, and indeed the model itself, may have been checked out of the project analysis parameter database (PAPD) by the user since he finds them useful in some way. Any of the members of a project's analysis group who has the necessary access rights can check a parameter out of the PAPD. As a result the models and their components have to be managed within the context of the project. When checking a parameter out of the PAPD a member can select only a specific version of the parameter for use in a particular model, as explained in section 7.2. Consequently the persistent identifier of the parameter is unique within the member's local model and it is used to manage the objects, and their associations, within the local address space of the user's application in the same way this was done in the single user application described in chapter 4. Within the context of the project, the management of objects is based on the use of two entities, namely the object's persistent identifier and its version number. The combination of the persistent identifier and the version number is called the selection identifier, which is unique within the scope of the project[1]. For project-wide management of objects, the collaboratory services make use of the selection identifier (where applicable).

Services: The collaboratory services support both the explicit and implicit collaborative actions that take place during worksessions. Once a user has successfully logged in to work on a project, he can use these services. The different services are configured as described in sections 7.3.2 through 7.3.5, and each comprise of objects on the project server and on the members' workstations. The service-objects on the members' workstations have reserved identifiers and are managed within the local processes. It is the task of the project manager to ensure that the collaboratory service-objects are properly installed and running on the project server.

[1] see section 6.2

9.2.2 Class App(lication)

This class manages objects in the local address space of the workstation in exactly the same way as in the single user application described in section 4.3.3. The HashMap `objectMap` of class `App` also contains the reserved identifiers and references of the local collaboratory service-objects.

9.2.3 Class AppObject

Class `AppObject` is the super class of all parameters of the finite element analysis, i.e. the model, its equation, as well as the nodes, locals, supports, materials, elements, nodeloads, lineloads, and arealoads. Whenever any of these objects are instantiated in the local address space of a workstation, the constructor `AppObject(identifier)` enters the object in the object map of class `App`(lication). This is similar to the single-user application.

The use of analysis parameters within the distributed setting of a project places demands on the parameters that were not required in the single-user application. Project analysis parameters need additional attributes and functionality, and those listed below were added at the superclass level, i.e. to class `AppObject`:

- `version`: instance of class `Version`. A `version` has an integer version number, the versioning history as a list of previous version numbers, and a time stamp indicating the time the version number was allocated.
- `createNewVersion()`: method which is executed to update the version attribute of the parameter.
- `femHasChanged`: boolean which is true when the parameter has undergone fundamental change, false otherwise.
- `usedInConsistentMode`: boolean which is true when the parameter is being used in consistent mode, false otherwise.
- `femChangeSupport`: instance of class `FemChangeSupport`[1], responsible for propagating fundamental changes made to parameters that are used in consistent mode.
- `updating`: boolean which is true while a parameter used in consistent mode is updating itself in response to a change reported through the consistency service, false otherwise.
- `aOwriteExternal()` and `aOreadExternal()`: methods called by parameters when they are written to or read from byte streams during network transmission. These methods are responsible for writing/reading the two non-transient attributes at the `AppObject` level, namely the persistent identifier string and the version.

[1] see section 9.5.3.1

9.3 Implementation of the model components

The component objects of a finite element model perform the functions dictated by the finite element theory. For a thin plate model, the component classes, and their attributes and functionality are described in section 4.4. The distribution of project analysis parameters places additional demands on the model components, as described in chapter 7. Meeting the demands of distribution is achieved by extending the component classes as described below.

Attributes: Additional attributes required for the distribution of parameters in the collaboratory are the same for all the components. Consequently they were added to the superclass of all components, i.e. `AppObject`, as described in section 9.2.3 above.

Functionality: Updating the version of a component is taken care of at the superclass level as described in section 9.2.3 above. Additional functionality required for distribution depends on the particular component class, and is provided through the implementation of two interfaces, namely `Externalizable` and `FemChangeReporter`:

Interface Externalizable: The Java interface `Externalizable` is part of the `java.io` package, and is implemented by classes that wish to have complete control over the format and contents of the streamed format of its objects. In this implementation the streams are network socket streams, and the purpose of implementing the interface is to ensure that no redundant data has to be transported over the network. `Externalizable` specifies two methods:

- `void writeExternal(ObjectOutput out)`: Write object data to the outputstream. The fundamental data that has to be written for each component class is listed in table 7-1.
- `void readExternal(ObjectInput in)`: Read object data from the input stream.

Interface FemChangeReporter: Methods of this interface are used for two purposes, namely versioning and consistency control.

- ***Versioning:*** For versioning purposes, `FemChangeReporter` specifies the following two methods:
  - `void setFeChangeInd(boolean value)`: This method is used to set the value of the boolean indicator which is used to register that fundamental change has occurred in a parameter.
  - `boolean getFeChangeInd()`: This method returns true if the parameter has been fundamentally changed since the previous version of the parameter was created, false otherwise. For newly instantiated parameters it returns true. When a set of parameters is traversed for versioning purposes, this method indicates whether a parameter requires a new version.

- ***Consistency:*** Parameters that have been checked out of the project analysis parameter database for use in consistent mode have to report to the collaboratory's consistency service when they are subjected to a fundamental change. Furthermore, they have to be able to update their attributes using information received from the consistency service regarding a remote change to the parameter. The two methods below are aimed at providing these functionalities:
  - `void reportFemChange(String methodName, Object[] oldValues, Object[] newValues)`: This method ensures that a fundamental change to a consistent parameter is reported[1]. Any changes to parameters are made through `set`-methods, and the method's name and old and new attribute values are reported.
  - `void update(String methodName, Object[] newValues)`: This method ensures that a consistent parameter is capable of changing attributes to the given new values by calling the appropriate `set`-method. The change that occurs during an update is not reported to the consistency service since this would cause a feedback loop.

9.4 Implementation of the model

An object of class `Model` is used to manage the component objects of the finite element model as described in section 4.5. The analysis algorithms are executed with the aid of an instance of class `Analysis`. In the process an equation (instance of class `Equation`) is added to the model, making the model a complete unit of information as described in section 5.2.1. The extentions required for using models in the distributed collaboratory are described below.

9.4.1 Class Model

Class `Model` is a subclass of `AppObject`, and as such inherits the attributes and functionality described in section 9.2.3 above. Furthermore, since models are project analysis parameters, they have to implement the interfaces `Externalizable` and `FemChangeReporter`.

Interface Externalizable: The model is a set of sets of components, and its streamed format does not contain the components themselves, but their selection identifiers, as explained in section 6.2.

- `void writeExternal(ObjectOutput out)`: When a model is written to an object stream, its component sets are traversed. For each set its name is written to the stream (as a String), followed by the selection identifier of each component of the set.
- `void readExternal(ObjectInput in)`: Reading a model from an object input stream involves reading the set names and corresponding selection identifiers of the components. The selection identifiers are stored in the repective component sets. If the

[1] see section 9.5.3

components themselves are required, the sets are traversed and the selection identifiers are used to retrieve the corresponding component objects.

Interface FemChangeReporter: This interface is used in the versioning and consistency control of the model.

- ***Versioning:*** For versioning purposes, a fundamental change to any of a model's components, or the addition or removal of a component object represents a fundamental change to the model and is registered as such using the interface method `setFeChangeInd(true)`. User controlled versioning would typically take place at the point in time when a user considers a model to be complete and useful. For this purpose a method which traverses the component sets was added to allow versioning of the model and all its components in one operation. The model itself requires a version update when the interface method `getFeChangeInd()` returns `true`.
- ***Consistency:*** When a model is checked out of the project analysis parameter database for use in consistent mode, all its components are used in consistent mode as well. The components handle their consistency responsibilities themselves, as described in section 9.3 above. Consequently the model is responsible only for consistency actions pertaining to the addition and removal of components to and from the model. The interface methods are implemented as follows:
  - `void reportFemChange(String methodName, Object[] oldValues, Object[] newValues)`: Components are added to the model with the different `add`-methods (e.g. `add(IcNode node)`), and removed using the respective `remove`-methods (e.g. `removeNode(String pid)`). The appropriate one of these method names will be the reported `methodName`. The parameter `oldValues` is always `null`, while `newValues` is either the component that was added, or the identifier of the component that was removed.
  - `void update(String methodName, Object[] newValues)`: The model simply adds or removes a component as dictated by the `methodName` and `newValues`.

9.4.2 Class Analysis

An object of class `Analysis` is used to analyse a model as described in section 4.5.2. If required, it instantiates an object of class `Equation` and adds it to the model. Class `Analyis` is purely a functional class, and its instance is not a project analysis parameter. It exists only in the address space of a member's workstation, and has no further relation to the collaboratory.

9.4.3 Class Equation

An instance of class `Equation` is used to form the finite element system equations, to decompose the coefficient matrix, and to solve the equations as described in section 4.5.3. In order to make the results of a model available to other members of the collaboratory, the

equation is treated as a project analysis parameter. Consequently it extends class `AppObject`, and implements the interfaces `Externalizable` and `FemChangeReporter`.

Interface Externalizable: This interface is used to manage the streamed format and content of the equation.

- `void writeExternal(ObjectOutput out)`: The Cholesky factor of the system matrix, and the primal and dual solution vectors are the valueable attributes of an equation. However, the volume of the Cholesky factor is a problem when network tranport is involved, as explained in section 5.2.9. Consequently only the primal and dual solution vectors are present in the streamed format of an equation.
- `void readExternal(ObjectInput in)`: The primal and dual solution vectors are read from the input stream.

Interface FemChangeReporter: This interface is used only for versioning purposes. Since the equation is formed and processed by the analysis procedures, it cannot be used in consistent mode.

- ***Versioning:*** Any change to an equation is fundamental, and is registered as such using the interface method `setFeChangeInd(true)`. During the versioning process an equation requires a version when the interface method `getFeChangeInd()` returns `true`.
- ***Consistency:*** Since the equation cannot be used in consistent mode both the methods `reportFemChange` and `update` perform no action.

9.5 Implementation of the collaboratory services

The need for collaboratory services is described in section 5.3.2 and specifications regarding their functionality are made. More details regarding their architecture and interaction with the project analysis parameters are explored in section 7.3. Implementation aspects of the services are described in this section.

9.5.1 Project analysis parameter database service (PAPD service)

The configuration of the PAPD service shown in figure 7-2 comprises of three parts, namely the project analysis parameter database (PAPD), the PAPD server, and the PAPD client.

9.5.1.1 Project analysis parameter database (PAPD)

In section 6.5.2.3 it was argued that relational database technology would be preferable for the PAPD. The database design has to support the storage of project analysis parameters, and could be kept extremely simple since the application takes care of establishing the relations between parameters. The database tables and storage schemes are described below.

Parameter tables: For each class of project analysis parameter a database table is available for storing instances of the class. Instances are stored row-wise in the table, and a table column is provided for each stored attribute. Certain columns are common to all the parameter tables, while others provide for the particular attributes of the corresponding parameter. The table names are listed in table 9-1.

Database table name	For storage of parameters of class:
tNodes	`Node`
tLocals	`Local`
tSupports	`Support`
tMaterials	`Material`
tElements	`Element`
tNodeLoads	`NodeLoad`
tLineLoads	`LineLoad`
tAreaLoads	`AreaLoad`
tEquations	`Equation`
tModels	`Model`

Table 9-1: Database table names

Common variables: Two fundamental attributes are common to all analysis parameters, namely the persistent identifier and the version. In addition, for access control purposes the database contains information about each parameter's owner, the taskgroup he belongs to, and the access code of the parameter. The variables that are present in all parameter tables are listed in table 9-2.

Table column name	Variable
pid	Persistent identifier
versionNumber	Version number
versionTime	Version timestamp
versionHistory	List of previous versions
ownerId	Login username of owner
taskGroupId	Name of taskgroup
accessCode	Parameter access tag (see section 7.4)

Table 9-2: Variables present in all database tables

Particular variables: The database table assigned to each class of analysis parameter has columns for storing the state of fundamental attributes that are particular to the parameter class. These attributes are listed in table 7-1. For the component parameters of a model, the table column names required to store their fundamental attributes are indicated in figure 9-1 below. The column names are self-explanatory when read in conjunction with section 4.2.

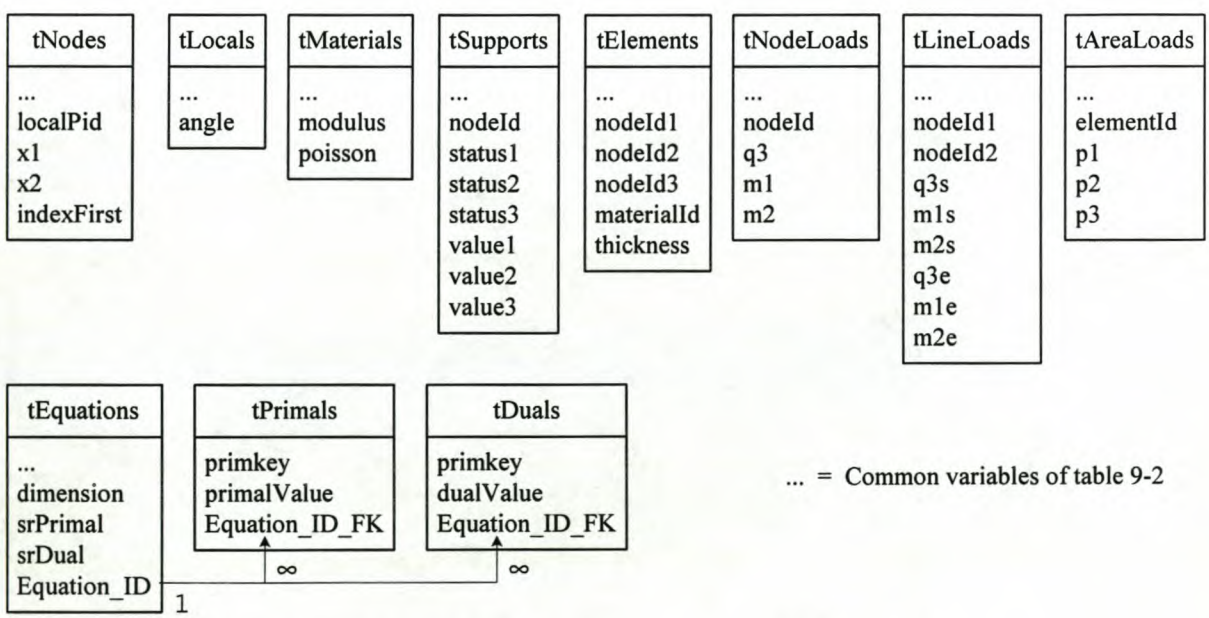


Figure 9-1: Model component parameters database tables

The storage mechanism for equations have to deal with the problem that the dimensions are not fixed, consequently values cannot be stored in a fixed number of columns. For this purpose the additional tables tPrimals and tDuals are introduced to store elements of the primal and dual solution vectors respectively. A specific equation, stored in table tEquations, use the boolean variables srPrimal and srDual to indicate whether the primal and dual vectors have been stored. The unique integer key, Equation_ID, which is auto-generated at storage time, is used to identify the particular equation's primal and dual elements by comparing it for equality with the Equation_ID_FK.

Model and component set tables: The storage of models entails the storage of the sets of selection identifiers of the model components[1]. The sizes of the component sets are not fixed and consequently the selection identifiers have to be stored row-wise. The storage structure for models is shown in figure 9-2. For each component class a Sets-table is used to store the selection identifiers of the component, e.g. tNodeSets is used to store `Node` components. When a model is stored, a unique integer Model_ID is auto-generated, which is put equal to the Model_ID_FK in the Sets-tables to identify the particular model's components.

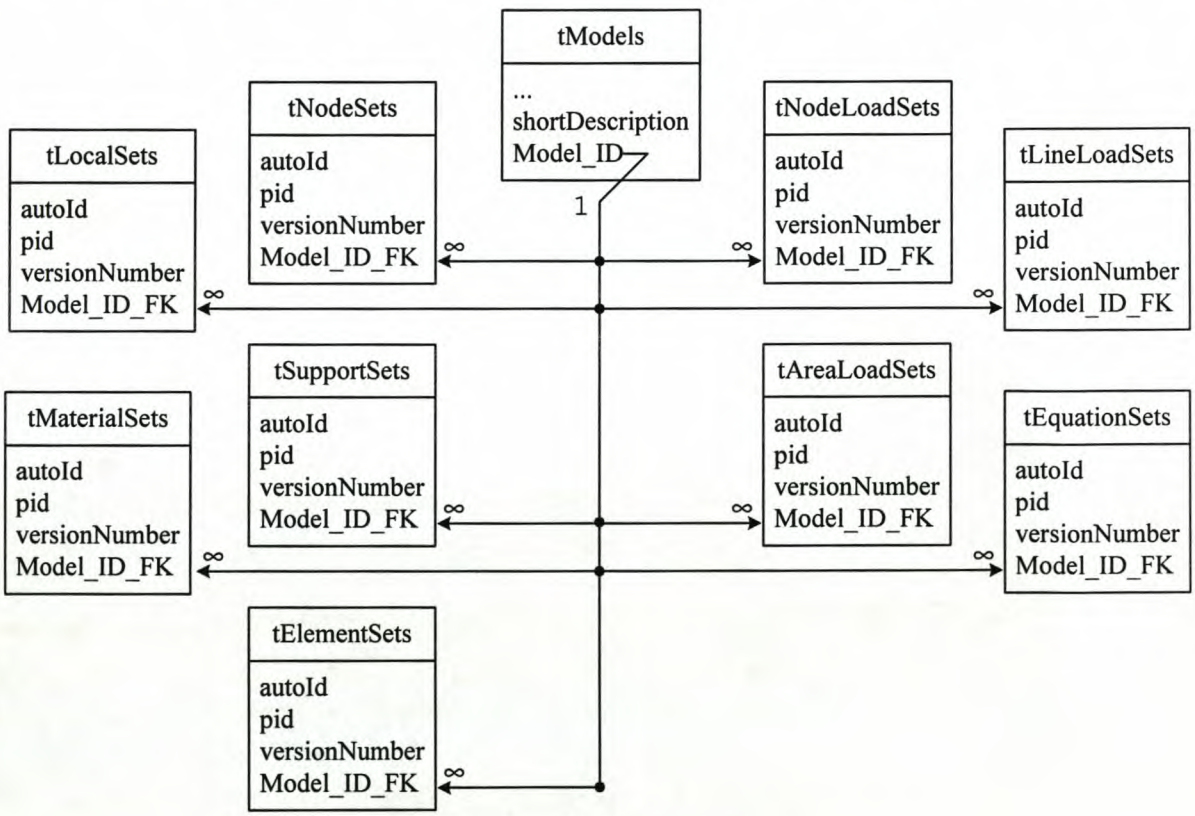


Figure 9-2: Storage structure for models

[1] see section 6.2

9.5.1.2 PAPD server

The PAPD server is available on demand on the project server to enable members of the analysis group to check parameters into and out of the project analysis parameter database.

Classes overview: The classes comprising the PAPD server are depicted in figure 9-3. The server is an instance of class `PAPDserver` which accepts socket connections from clients at the workplaces. It then creates and allocates a `PAPDserverThread` to communicate exclusively with the client. In serving client requests, the `PAPDserverThread` communicates with a `DbObject`. The latter instructs the appropriate database representative, e.g. `DbNode`, to deal with requests pertaining to `Node` parameters. The database representatives use a `PAPDcomm` instance to communicate with the project analysis parameter database (PAPD). This communication is based on the Structured Query Language (SQL) through Java's JDBC API [White 1999].

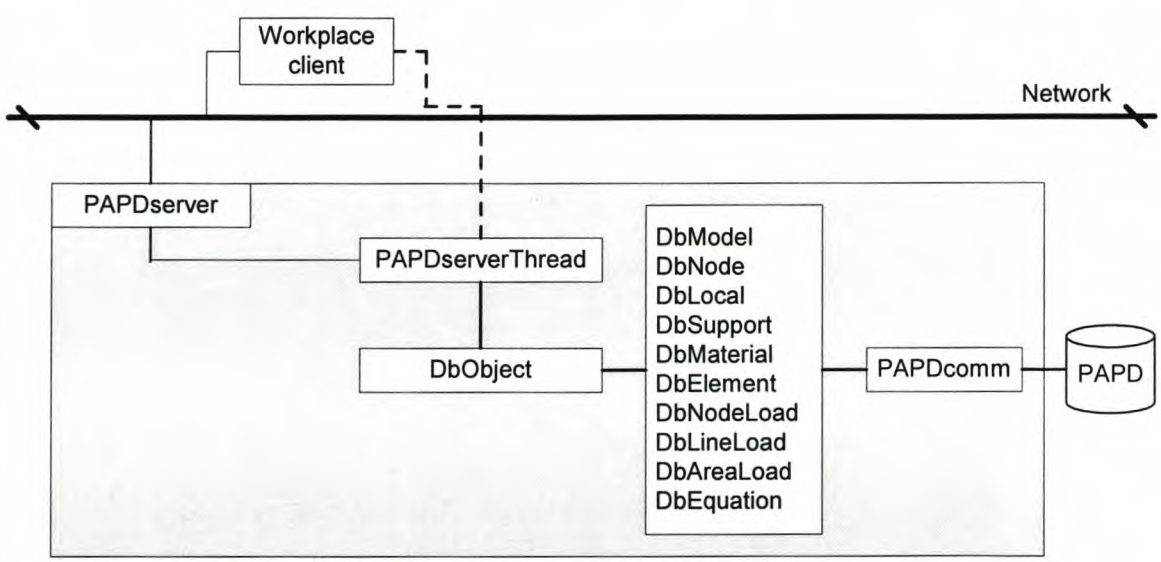


Figure 9-3: Classes comprising the PAPD server

Class PAPDserver: An object of this class act as the PAPD server, and is constructed as follows: `PAPDserver(projectId, portNumber, centralConsistency)`.
The project identifier is used for control purposes and the `portNumber` is the port on which the server accepts connections from clients. The central consistency service[1] is notified when a parameter is checked out for use in consistent mode. When a client establishes a connection, the `PAPDserver` creates a `PAPDserverThread` for the client, and further communication takes place between the client and its dedicated thread.

Class PAPDserverThread: For each workplace client, an instance of this class is created by the `PAPDserver`, as follows: `PAPDserverThread(socket, papdServer)`

[1] see section 9.5.3.2

The PAPDserverThread instance is bound with the socket connection to the workplace client, and serves that client through the following protocol:

- The initialization command init has to be followed by the following strings: user name, taskgroup, database url, database username, and database password. The database url, username and password are used to establish a connection to the project analysis parameter database (PAPD), which is managed by a DbObject instance (see Class DbObject below). The reply completed is sent after successful initialization.
- The command put is used to place a parameter in the PAPD. It is followed by the parameter's access tag and the streamed parameter[1], which are handed over to the DbObject for storing. The reply is true or false, depending on whether the storage attempt was successful or not.
- The command get is used to check a parameter out of the PAPD. It is followed by the parameter's persistent identifier string, its integer version number, and the mode (boolean) of checkout of the parameter, true for consistent mode and false for free mode. The request is handled by the DbObject, and the reply is either the streamed parameter, or the string null.
- The command close is used to close the database connection and terminate communication.

Class DbObject: An instance of this class is used by the server thread to manage communication with the PAPD.
It is created as follows: DbObject(dbURL, dbUserName, dbPassword)
Class DbObject has information about the PAPD structure as described in section 9.5.1.1 and has representatives of all the analysis parameters in order to deal with their specific storage and retrieval requirements. Class DbObject's key methods are:

- boolean put(appObject, ownerId, taskgroupId, accessTag): This method attempts to store an AppObject, i.e. any analysis parameter, and should be successful unless the parameter is already present in the PAPD. It returns true if the storage attempt was successful, false otherwise.
- AppObject get(persistentIdentifier, versionNumber, userId, taskgroupId, cfMode): This method attempts to retrieve a parameter identified by its persistent identifier and version number from the PAPD. When cfMode is true, the parameter is checked-out in consistent mode, otherwise in free mode. The return value is either the requested parameter, or null if the parameter could not be retrieved.

Classes DbModel, DbNode, DbLocal, DbSupport, DbMaterial, DbElement, DbNodeLoad, DbLineLoad, DbAreaLoad, DbEquation: Instances of these classes assist the DbObject in the storage and retrieval of analysis parameters of the respective class that each of them represent. The DbObject selects the appropriate representative instance,

[1] see description of interface Externalizable in sections 9.3 and 9.4

and delegates the `put` and `get` responsibility to it. The representatitve generates the necessary SQL statement, and uses the `PAPDcomm` instance to execute a query, or an update of the database.

Class PAPDcomm: An instance of this class is created by the DbObject to provide the necessary communication with the PAPD.
It is constructed as follows: `PAPDcomm(dbURL, dbUserName, dbPassword)`
The constructor parameters are used to establish a connection to the database indicated by the URL and made accessible by its user name and password. Key methods of class `PAPDcomm` are:

- `ResultSet executeQuery(String querySQL)`: This method returns the result set generated by executing the given SQL query.
- `boolean executeUpdate(String updateSQL)`: An attempt is made to update the database using the given update-SQL statement. The method returns `true` if the update was successful, `false` otherwise.
- `boolean closeConnection()`: Close the connection to the database.

9.5.1.3 PAPD client

The PAPD client is part of the application software that is installed at each of the workplaces. It requests services from the PAPD server according to the user's instructions. The available services are described in the previous section. In this implementation the user can check an analysis parameter of any class into, or out of the PAPD. Parameters can be checked out for use in free mode or consistent mode. The PAPD client consists of a single class.

Class PAPDclient: An instance of this class is created by a member of the analysis group, who has successfully logged in to work on one of the project's tasks, when he starts the PAPD service. It is created as follows: `PAPDclient(userName, taskgroupId, project)` : The user name, taskgroup and project are known from the user's login, and are required for registering with the server. Information like the server host, port number, and the database URL, user name and password are contained in a properties object of the `project`. During instantiation the `PAPDclient` uses the `init` protocol of the `PAPDserverThread` described in the previous section. The methods of the `PAPDclient` correspond to the commands of the `PAPDserverThread` protocol:

- `boolean put(appObject)`: This method communicates with the server in order to check a project analysis parameter into the PAPD. If the attempt is successful the return value is true, false otherwise.
- `AppObject get(selectionIdentifier, consistent)`: The method communicates with the server in order to check a parameter, identified by its selection identifier, out of the PAPD. The checkout mode is either consistent (true) or free. The return value is either the parameter, or the string "null".
- `boolean closeConnection()`: Used to terminate the communication with the server.

9.5.2 Persistent identifier service (PID service)

The PID service has a standard client-server configuration, with the PID server installed on the project server, and a PID client on each of the workstations at the collaboratory workplaces.

9.5.2.1 PID server

The PID server is available on demand on the project server and has the task of generating persistent identifiers for a project's analysis parameters. It comprises two classes:

Class PidServer: An object of this class acts as the PID server and is constructed as follows: `PidServer(projectId, portNumber)`: The project identifier is used for control purposes, and the port number is the port on which the server accepts connections from clients. Class `PidServer` creates identifiers according to the naming schema Task.User.Component.Number described in section 6.1. Towards this purpose the PID server maintains a tree with the project identifier at its root, branching to task identifiers, to user identifiers, and finally to component class names with the last used number as its leaf. When a client establishes a connection, the `PidServer` creates a `PidServerThread` for that client and further communication takes place between the client and its dedicated `PidServerThread`.

Class PidServerThread: For each client an instance of this class is created as follows: `PidServerThread(pidServer, clientSocket)`: The `PidServerThread` serves the client through the socket connection according to the following protocol:

- The command `getPid`, followed by the task identifier, the user identifier, and the component class name is sent to retrieve an identifier. The reply is the next available identifier according to the naming schema mentioned above.
- The command `close` is used to terminate communication.

9.5.2.2 PID client

The PID client is a component of the distributed analysis application installed at each workstation and its task is to maintain a cache of identifiers in a local buffer to ensure fast access to identifiers during interactive worksessions. The PID client consists of two classes:

Class PidCache: An instance of this class is created when a member starts the PID service after logging on to work on a project. The instance is created as follows: `PidCache(taskId, userId, project)` : The task identifier, user name and project are known from the login step. During instantiation the `PidCache` attempts to read previously cached identifiers from a disk file which is created each time a worksession is terminated. The PID server is not contacted right away. However, when this becomes necessary the project server host name and port on which the PID server listens for client connections are obtained from the `project`. A `PidCache` contains a hash map that maps component class names to lists of identifiers for the corresponding component types. The sizes of these iden-

tifier lists are configurable. Access to the identifiers is provided by the method `getPid(compClass)`, which returns an identifier for the specified component class. A `PidCache` is a low priority thread which constantly monitors the number of identifiers in the identifier lists. Whenever more than a quarter of the configured number of identifiers have been used, another low priority thread, namely an instance of class `CacheUpThread` is created to retrieve identifiers.

Class CacheUpThread: An instance of class CacheUpThread is created as follows:
`CacheUpThread(pidCache, getMap)`: The `CacheUpThread` instance contacts the PID server which instantiates a `PidServerThread` to serve it. The `getMap`-parameter specifies the number of identifiers required to fill up the identifier list for each component class to its configured size. Using the `getPid`-command of the `PidServerThread`, the `CacheUpThread` retrieves the required number of identifiers for each component class and adds them to the `pidCache`. Upon completion it closes the connection using the `close`-command of the server protocol.

9.5.3 Consistency service

Configuration overview: The consistency service has a peer-to-peer configuration with the central consistency service installed on the project server and the workplace service on each of the workstations of the collaboratory. The classes comprising the consistency service are shown in figure 9-4. The central consistency service is an instance of class `CentralConsistency`. It accepts socket connections from the workplace consistency services, which are instances of class `WorkplaceConsistency`.

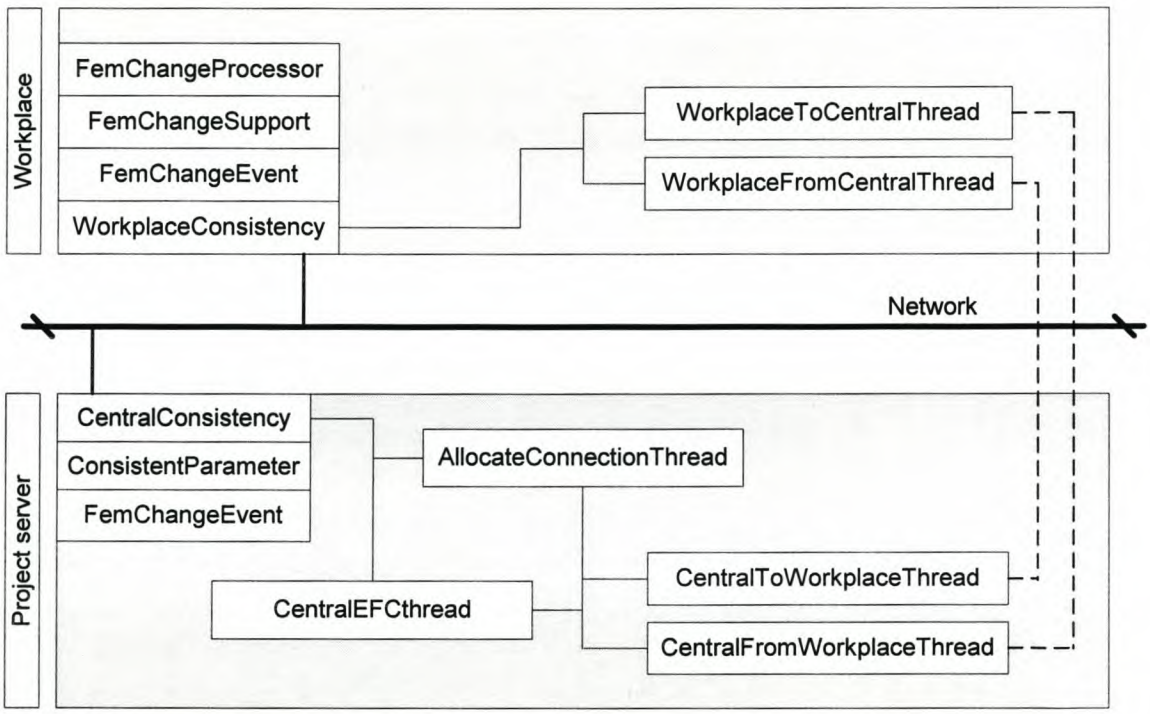


Figure 9-4: Classes comprising the consistency service.

9.5.3.1 Workplace consistency service

Class WorkplaceConsistency: An instance of this class manages the workplace consistency service. The instance is created when a member who has logged on to work on the project starts the consistency service, as follows: `WorkplaceConsistency(userName, project)` : The user name and project are known from the login procedure. Properties of the project specify the host name and port through which a connection to the central consistency service can be established. During instantiation an attempt is made to recreate the state of the service from a backup disk file. The state of the `WorkplaceConsistency` is contained in two lists, namely the list of changes still to be forwarded to the central service, and the list of unprocessed changes that have been received from the central service. The changes are instances of class `FemChangeEvent`. The forwarding and receiving of changes

are performed by two threads that are spawned by the `WorkplaceConsistency`, respectively the `WorkplaceToCentralThread` and the `WorkplaceFromCentralThread`.

Classes FemChangeEvent, FemChangeSupport: All changes are distributed as instances of class `FemChangeEvent`. The analysis parameters implement the interface `FemChangeReporter` described in section 9.3. Herein they are assisted by instances of class `FemChangeSupport`. The `FemChangeSupport` creates the `FemChangeEvent` instances and forwards them to the registerded `FemChangeListener`, in this case the `WorkplaceConsistency`, as follows:

- `FemChangeEvent(sourceSID, methodName, oldParameters, newParameters)` : The selection identifier of the analysis parameter on which a fundamental change was made, the name of the method that effected the change, and the method's previous and new parameter values[1] are specified. The method name and its new parameters are used to perform an update of the parameter at remote locations that receive the change, while the old parameter values are used to aid the receiving member in deciding whether to accept or reject the change.
- `FemChangeEvent(sourceSID, newVersion)` : This form is used when a new version of a consistent mode analysis parameter is created. The `newVersion` is the complete, newly versioned `AppObject`. When a receiving member accepts this change, the complete new version of the analysis parameter is installed in his workspace.

Class WorkPlaceToCentralThread: An instance of this class is a low priority thread that monitors the list of changes that have to be forwarded to the central consistency service. When the list is not empty, the thread sequentially forwards each change and removes it from the list upon confirmation from the central service that the change has been registered there. As a result local changes are buffered until the workplace service is started and this thread becomes active.

Class WorkplaceFromCentralThread: An instance of this class is a low priority thread that waits for changes from the central consistency service. As soon as it receives a `FemChangeEvent`, it is added to the list of unprocessed changes, and the central service is sent a confirmation of its receipt.

Class FemChangeProcessor: An instance of this class is used to deal with the unprocessed changes in the remote-changes list of the `WorkplaceConsistency`. The member considers the old and new values of the parameter, and has the opportunity to either accept or reject the change. If the change is accepted, the data required by the `update`-method of interface `FemChangeReporter` is extracted from the change and the matching analysis parameter's `update`-method is called.

[1] see section 9.3

9.5.3.2 Central consistency service

Class CentralConsistency: An instance of this class act as the central consistency service and is constructed as follows: `CentralConsistency(projectId, portNumber)`: The project identifier is used for control purposes and the port number is the port on which the server accepts connections from the workplace consistency services. The `CentralConsistency` holds a map of parameters used in consistent mode. The keys of the consistent-use map are the selection identifiers the parameters, and the mapped values are related instances of class `ConsistentParameter`.

Class ConsistentParameter: An instance of this class is created when a parameter is checked out of the PAPD for use in consistent mode[1], provided such an instance does not already exist for the parameter, as follows: `ConsistentParameter(selectionIdenti=fier).` : A `ConsistentParameter` maintains a list of those members of the analysis group that have checked-out the related analysis parameter in consistent mode, as well as a list of all fundamental changes that were made to it by these members. The changes are instances of class `FemChangeEvent`.

Classes AllocateConnectionThread, CentralToWorkplaceThread, CentralFromWorkplaceThread : Whenever a workplace consistency service is started, it contacts the central service in order to establish socket connections between the two services. In reaction to such a request the `CentralConsistency` creates an `AllocateConnectionThread` which establishes two dedicated socket connections to the requesting workplace:

- An instance of class `CentralToWorkplaceThread` is created. This thread is connected to the `WorkplaceFromCentralThread` of the workplace service. Remote changes are forwarded from the central service to the workplace via this connection.
- An instance of class `CentralFromWorkplaceThread` is created. This thread is connected to the `WorkplaceToCentralThread` of the workplace service, through which workplace changes are forwarded to the central service. Upon receipt of a change a confirmation message is sent to the workplace. Connections between the workplace and the central service is terminated when the String "close" is sent from the workplace.

Class CentralEFCthread: An instance of this class is a low priority Event-Forwarding-Control thread that continuously monitor both the consistent-use map of the `CentralConsistency`, and the list of connected members. Its task is to identify all unprocessed `FemChangeEvents` on the consistent parameters that a connected member uses, and to have them sent to the member with the aid of the `CentralToWorkplaceThread`. Upon confirmation that the change has been registered at the workplace, the particular change is flagged as having been sent to the particular member. As a result changes are buffered on the central service until a member starts the workplace service.

[1] see section 9.5.1.2

9.5.4 Versioning service

The task of the versioning service is to facilitate the updating of versions of selected parameters by supplying new version numbers and timestamps to the workstations. The service has a standard client-server configuration, with the versioning server installed on the project server and the versioning clients at the workplaces.

9.5.4.1 Versioning server

The versioning server comprises of two classes, namely class `VersioningServer` and class `VersioningServerThread`.

Class VersioningServer: An instance of this class is the versioning server. It is constructed as follows: `VersioningServer(projectId, portNumber)` : The project identifier is used for control purposes, and the port number is the port on which the server accepts connections from versioning clients. During instantiation the `VersioningServer` attempts to restore its previous state from a backup disk file. The state of versioning is contained in a hash map where the persistent identifier of each versioned parameter is mapped to its last issued version number. When a client establishes a connection to the server, a `VersioningServerThread` is created to serve that client.

Class VersioningServerThread: An instance of this class is created as follows: `VersioningServerThread(versioningServer, clientSocket)` : The client is served through the socket connection according to the following protocol:

- The command `updateVersion`, followed by the persistent identifier of the parameter requiring a new version is sent to obtain a new version number and corresponding timestamp from the server. These two values are sent as the reply.
- The command `close` is sent to terminate communication.

9.5.4.2 Versioning client

The versioning client is a component of the distributed analysis application installed at each workstation. Its task is to communicate with the versioning server for the retrieval of versioning information.

Class VersioningClient: An instance of this class acts as the workplace's versioning client, and it is instantiated as follows: `VersioningClient(project)` : The server host name and port number, essential for establishing a connection to the server, are properties of the project. A connection to the server is established during instantiation. The `VersioningClient` provides the method: `boolean updateVersion(String pid)`. This method uses the protocol provided by the `VersioningServerThread` to retrieve the new version number and timestamp for the parameter with persistent identifier `pid`. It then updates the parameter's version. Communication with the server is terminated by sending the command `close`.

10 Examples and evaluation

A pilot distributed analysis application which provides a basic software infrastructure for supporting members of the analysis group in the execution of a set of collaborative analysis tasks was described in the previous chapter. In this chapter the execution of distributed analysis tasks is illustrated in two simple examples, and the proposed solutions for distributed analysis are evaluated in terms of the overhead resulting from working in a distributed environment.

10.1 Examples

Collaborative worksession: A user has the option of working in single user mode, or to work as a member of a project's analysis group. In single user mode the analysis task would be performed as described in chapter 4. In order to work as a member of an analysis group, the user has to start a collaborative worksession by logging in to work on the project, using his unique username and password[1]. He has to choose a task on which to work, the task having been assigned to the taskgroup of which he is a member. If the login was successful, the user will have access to the various services of the collaboratory, which enables him to execute the task while collaborating with other members of the analysis group, and specifically with members of the taskgroup.

Overview of an analysis task: In the single user environment, the geometry, materials, and static and kinematic boundary conditions of the analysis model (an instance of class `Model`) are specified by the user through the instantiation of new components of the model. A member of an analysis collaboratory has the additional option of using existing models and/or components that were made available in the project analysis parameter database (PAPD) by a collaborating team member. The components are instances of the classes `Node`, `Local`, `Support`, `Material`, `Element`, `NodeLoad`, `LineLoad`, and `AreaLoad`. After the necessary components have been instantiated, the analysis algorithms are executed, a system `Equation` is formed and solved, and the model's results are printed as described in section 4.2. New components are instantiated using the same text input file and format described in section 4.2, but their identifiers have to be obtained from the collaboratory's persistent identifier service[2]. If existing parameters are used, they have to be checked out of the project analysis parameter database with the aid of the PAPD service[3]. The user has the option to use an existing parameter either in free mode, or in consistent mode. In the latter case users have to deal with remote changes to the parameter that propagate through the consistency service. Fundamental changes made to parameters are registered by updating

[1] see section 7.3.1
[2] see section 7.3.3
[3] see section 7.3.2

the versions of the parameters. This action is controlled by the user and makes use of the versioning service[1] of the collaboratory.

10.1.1 Example 1: Basic modeling, storage and retrieval.

Layout: Consider a square plate of dimension 4x4[2], with thickness 0,15. It is loaded as shown in figure 10-1 with an areaload of intensity 2, diagonal lineloads of intensity 5, and a concentrated load in the centre of magnitude 10. The plate is simply supported along its outside edges, and a local system directed from point 2 to 4 is required and the centre point (for illustrative purposes). A feature of this example is that all the different analysis parameters are present in the model.

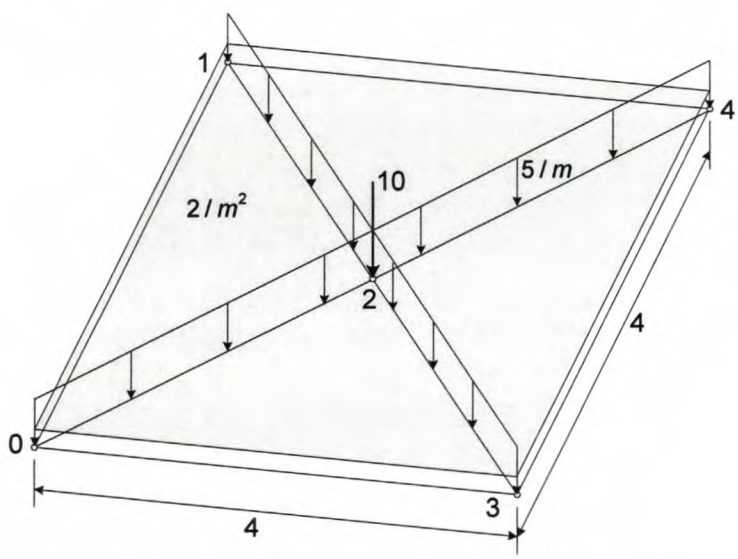


Figure 10-1: Layout of example 1

Project: The analysis of the plate is the project, called X1, which is to be executed by two engineers. One will model the geometry and kinematic boundary conditions, and the other will apply the loads and perform the analysis.

Tasks: Two tasks are identified, by name:

1. x1Geo: Modeling of the geometry and kinematic boundary conditions. This task is scheduled to be completed before the next one, and the geometric model generated here is to be used in the task that follows.
2. x1L&A: Loads modeling and execution of the analysis. The loading is simply added to the geometric model generated in task x1Geo, and the analysis algorithms are executed.

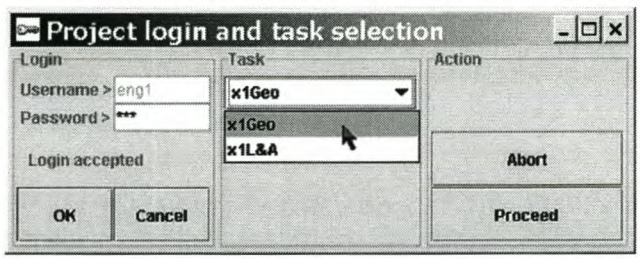
[1] see section 7.3.5
[2] Any consistent set of units may be used.

Taskgroups: Two taskgroups are identified, by name:

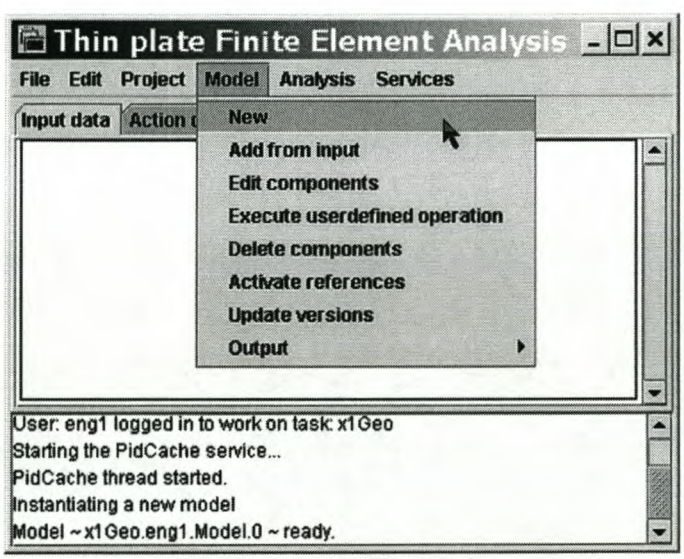
1. geoGrp: This group comprises one engineer with username eng1, and is responsible for task x1Geo.
2. l&aGrp: This group comprises one engineer with username eng2, and is responsible for task x1L&A.

Execution of task x1Geo: This task comprises the activities listed below, all of which are either explicitly or implicitly collaborative. These are:

1. Login to work on the project: The member uses his username (eng1) and password to log in and choose a task, after which he can use the collaboratory services.



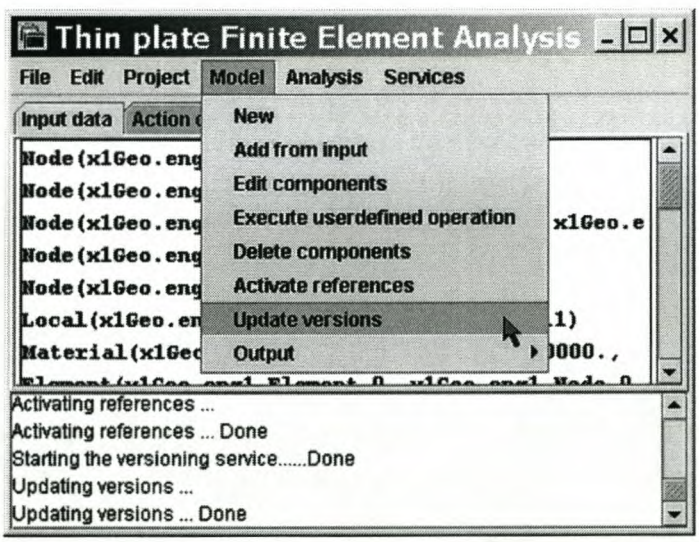
2. Create a model: Since all work takes place within the context of a model, a new model is created. The member, having logged in to work on the project, can start the PID service and define parameters using identifiers provided by the service. Identifiers are generated according to the scheme: task.user.class.number. The model's identifier is consequently x1Geo.eng1.Model.0.



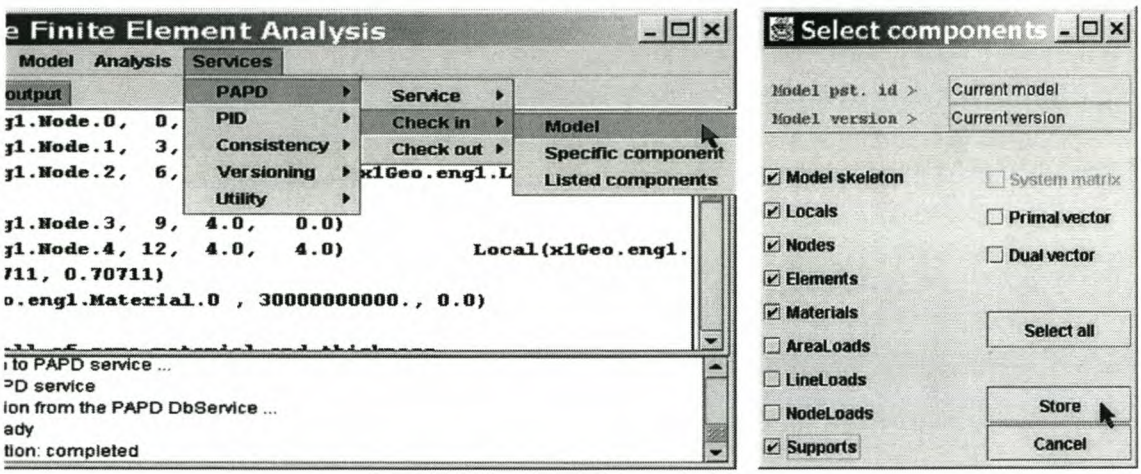
3. Add components to the model: The necessary geometric and kinematic boundary condition components are defined in the input area, and then added to the model. Examples of the input definition of various components are shown. The identifiers are those issued by the PID service.

```
Node(x1Geo.eng1.Node.2,  6,  2.0,  2.0,  x1Geo.eng1.Local.0)

Local(x1Geo.eng1.Local.0,  0.70711,  0.70711)

Material(x1Geo.eng1.Material.0,  30000000000.,  0.0)

Element(x1Geo.eng1.Element.0,
        x1Geo.eng1.Node.0, x1Geo.eng1.Node.2, x1Geo.eng1.Node.1,
        x1Geo.eng1.Material.0, 0.15)

Support(x1Geo.eng1.Support.0, x1Geo.eng1.Node.0, SIMPLE)
```

4. Version the model and its components: Once the engineer is satisfied that the model is useful, the analysis parameters are versioned using the versioning service. Initial version numbers are zero. Parameters cannot be checked into the project analysis parameter database (PAPD) if they have not been versioned.



5. Check the model and its components into the PAPD: Through this action the member makes parameters available to other members of the collaboratory, in this case to the engineer responsible for task x1L&A.



6. After the check-in step, the member should notify other members that the model is available, providing the necessary information for navigation, e.g. the model's identifier.

Database state: A snapshot of the PAPD at this stage is shown in figure 10-2. The model created in this task has the selection identifier x1Geo.eng1.Model.0:0 and is stored in table tModels. Its components' identifiers are stored in the various set-tables, e.g. tNodeSets, and the respective component parameters are stored in the class-tables, e.g. tNodes. The complete model, or any of its component sets, or any of its individual components can be checked out of the PAPD by other members with the necessary access rights.

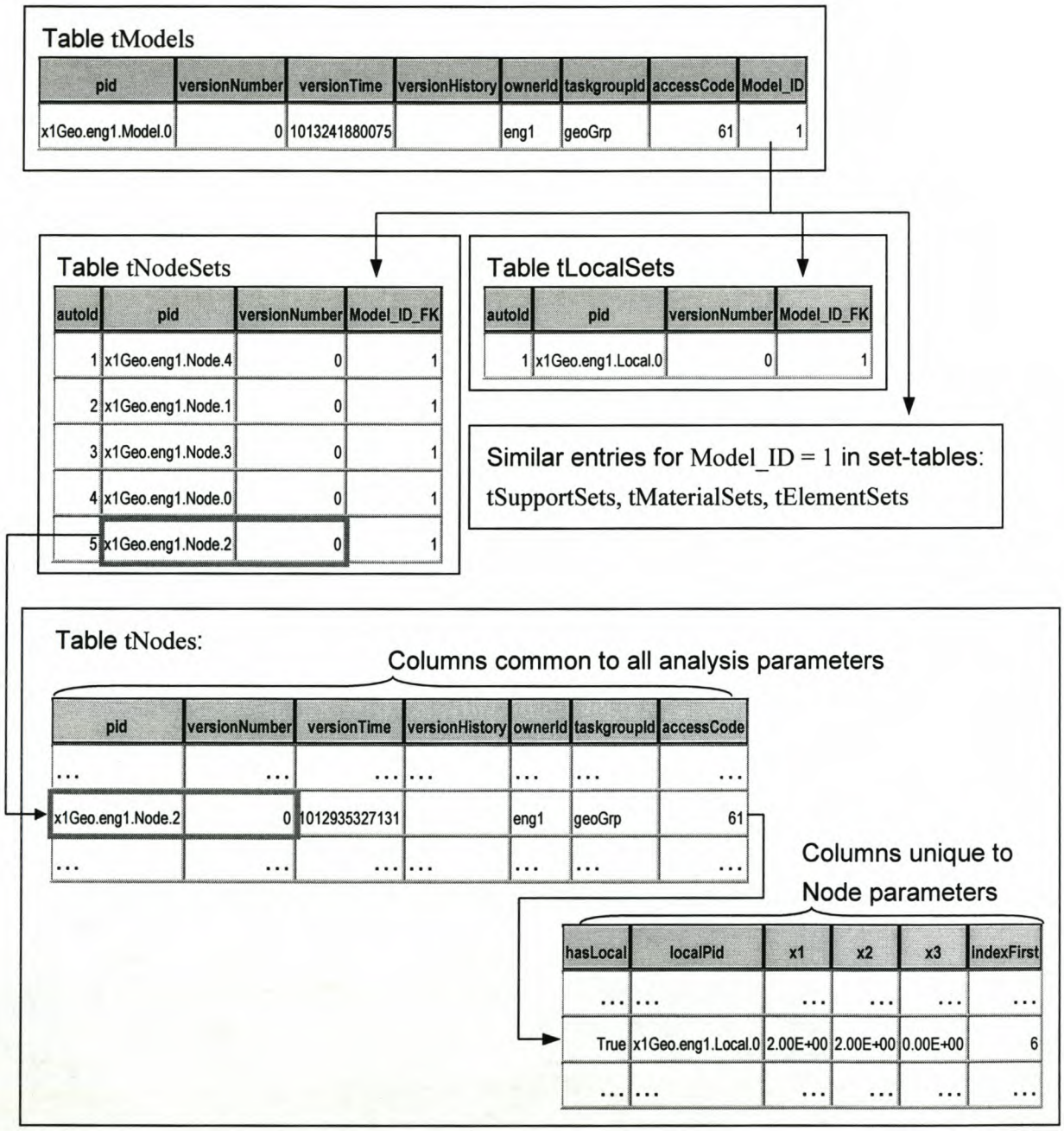
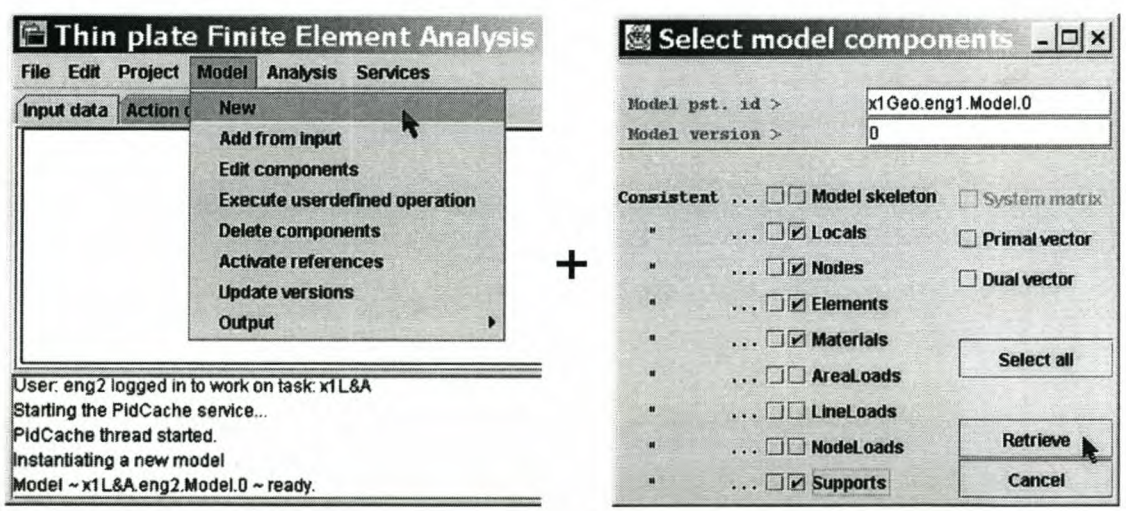


Figure 10-2: PAPD storage of model of task x1Geo.

Execution of task x1L&A: The task of applying loading and executing the analysis follows on task x1Geo, with the outright intention of using the analysis parameters created in that task. The activities performed by the engineer responsible for task x1L&A are the following:

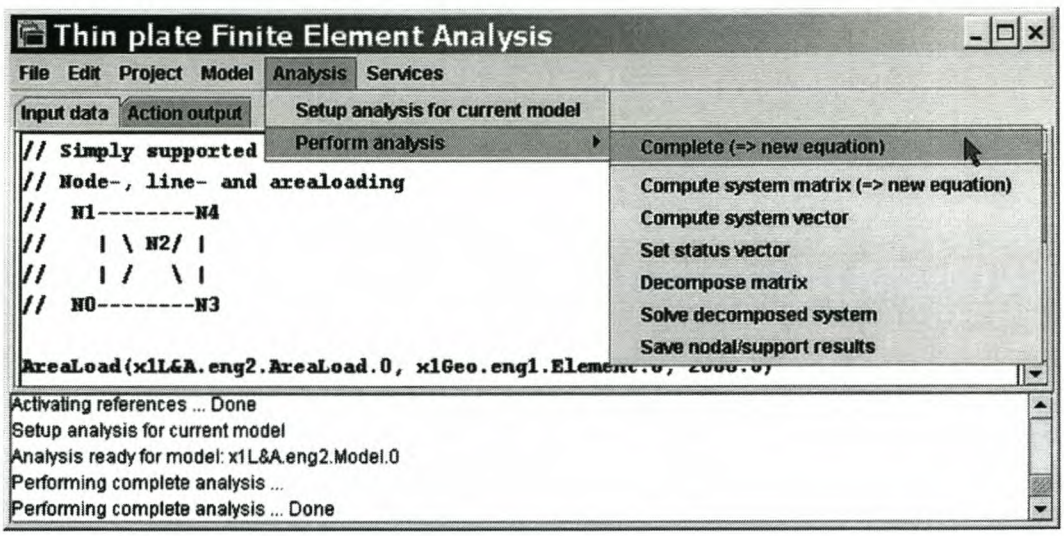
1. Login to work on the project: The member uses his username (eng2) and password to log in and choose a task, in this case x1L&A, after which he can use the collaboratory services.
2. Create the model context: Two possible courses of action may be taken, namely to create a new model and add the existing components ánd all new components to it, or to use the existing model, modify it by adding the required new components, and create a new version of it. Both these options are identical as far as functionality and storage requirements are concerned, and either project policy or personal preference would determine the choice. In this example a new model, with identifier provided by the persistent identifier service x1L&A.eng2.Model.0, is created (graphic below on the left), and the existing geometric components, i.e. the locals, nodes, elements, materials and supports of model x1Geo.eng1.Model.0:0 are checked out of the PAPD and added to it (graphic below on the right).



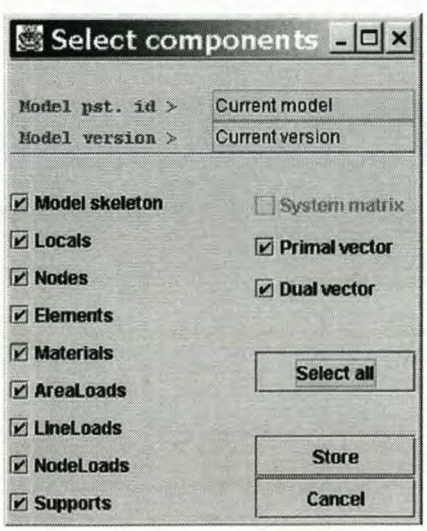
3. The loading components are subsequently provided by defining new components in the input area, and adding them to the model. Their identifiers are supplied by the persistent identifier service. The loading components refer to the persistent identifiers of the geometric components that they are applied to, as shown below:

```
NodeLoad(x1L&A.eng2.NodeLoad.0, x1Geo.eng1.Node.2, 10000.0, 0, 0)

LineLoad(x1L&A.eng2.LineLoad.0, x1Geo.eng1.Node.0, x1Geo.eng1.Node.2, 5000.0)

AreaLoad(x1L&A.eng2.AreaLoad.0, x1Geo.eng1.Element.0, 2000.0)
```

4. Execute the analysis algorithms: Once the necessary loading components have been defined and added to the model, and the connections amongst objects have been established (Activate references on the Model menu), model x1L&A.eng2.Model.0 can be analysed. In the process a new equation is created, with identifier x1L&A.eng2.Equation.0. The primal and dual vectors of this equation contain the analysis results.



5. Check the model and its components into the PAPD: Through this action the member makes parameters available to other members of the collaboratory, and he should notify them that the model is available, providing the necessary information for navigation, e.g. the model's identifier. In this case the model has been analysed, and other members may be interested in interpreting the results. During the check-in process the primal and dual vectors should be selected to be checked into the PAPD, as indicated in the graphic. It is important to note that the newly checked-in model will contain references to the selection identifiers of the geometric components that are already stored in the PAPD. These are not duplicated or rewritten. The loading components and result vectors, however, are entered into the PAPD for the first time.



Database state: A snapshot of the PAPD, indicating the storage of the model's equation components, is shown in figure 10-3. The model created in this task, with selection identifier x1L&A.eng2.Model.0:0, is added to table tModels. Its components' identifiers are stored in the various set-tables, which now include entries in the load-component sets tNodeLoadSet, tLineLoadSet, tAreaLoadSet, as well as an entry in the in the table for equations, i.e. tEquation-Sets.

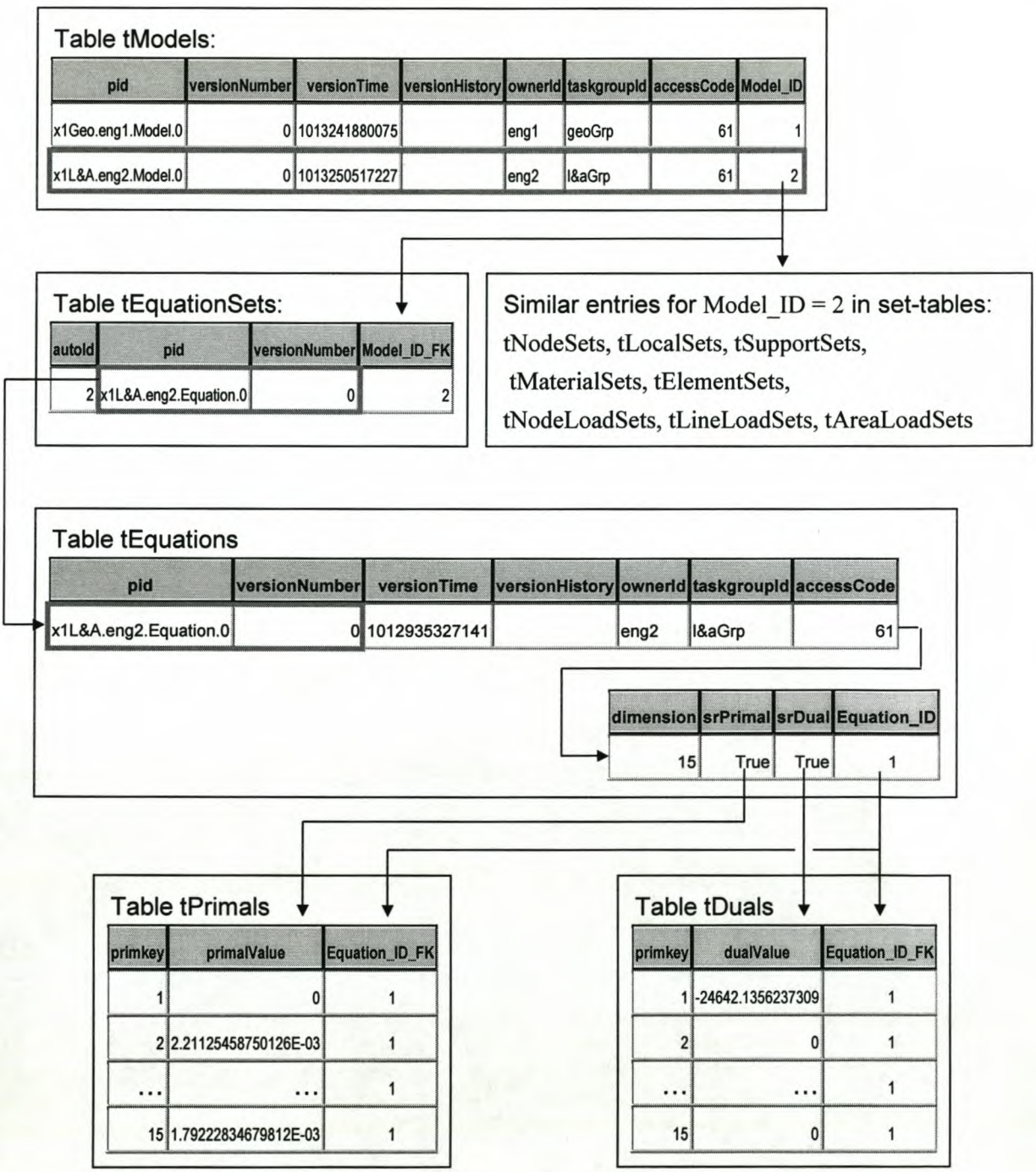


Figure 10-3: PAPD storage of equation created in task x1L&A.

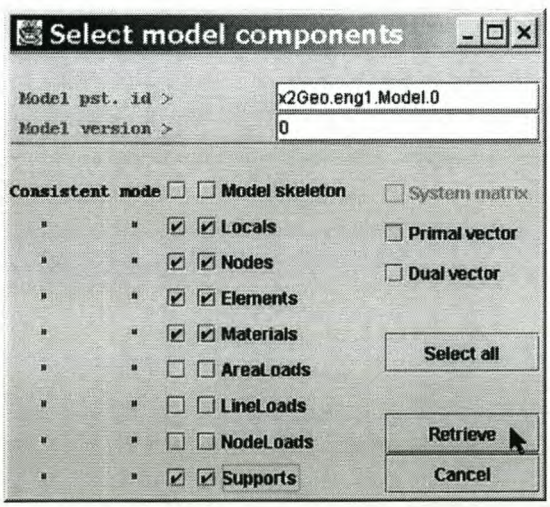
10.1.2 Example 2: Using parameters in consistent mode

The key functionality of the consistency service is demonstrated in this example.

Example setup: The same project, tasks and taskgroups as in example 1 above are used. To avoid ambiguity, the project is called X2, and the tasks x2Geo and x2L&A. The taskgroups and the engineers are named as in example 1, respectively geoGrp and l&aGrp, and eng1 and eng2. The geometric modeling of task x2Geo proceeds exactly as in example 1, up to the point where the parameters are checked into the PAPD. In the current example the parameters are checked into the PAPD with access codes that allow a member of a different taskgroup to use them in consistent mode. At the start of task x2L&A, the parameters are correspondingly checked out of the PAPD for use in consistent mode, whereas in example 1 they were checked out for use in free mode. Engineers may be working synchronously or asynchronously.

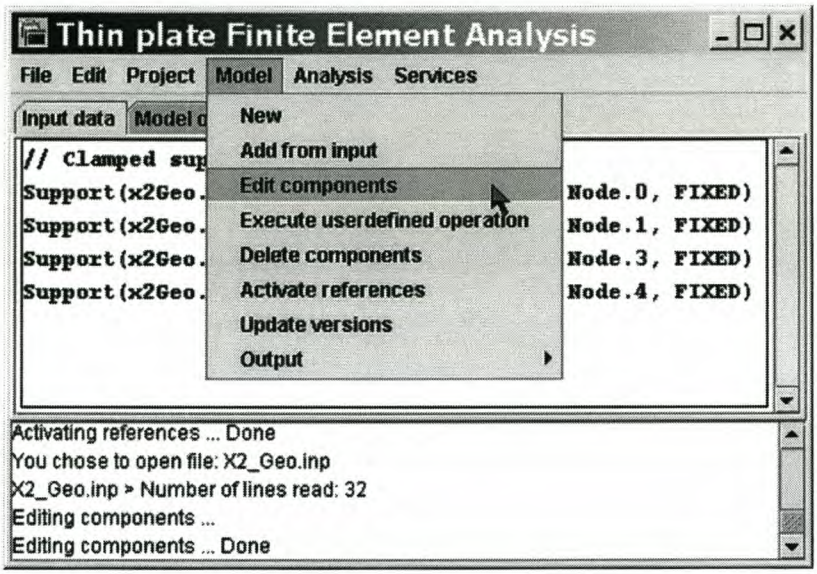
Execution of task x2L&A: The activities performed by the engineer responsible for task x2L&A are the following:

1. After logging in to work on the project and creating a new model (x2L&A.eng2.Model.0), the parameters created in task x2Geo are checked out for use in **consistent** mode.

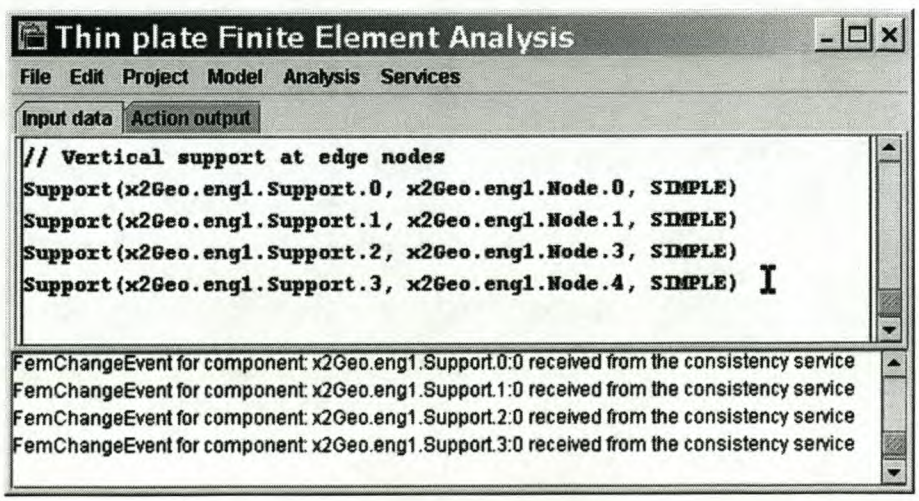


2. The loading components are subsequently provided by defining new components in the input area, and adding them to the model. The procedure is similar to that of example 1.

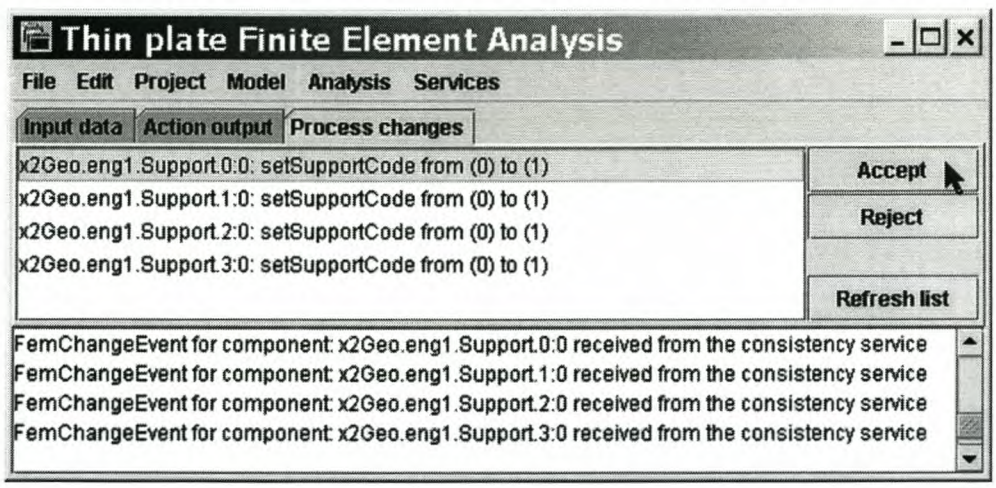
3. At this point in time the head of the analysis group informs engineer eng2 that the design of the plate has to be changed from simply supported to clamped support. The support components, created during the foregoing geometric modeling phase, are changed to "FIXED" as shown below.



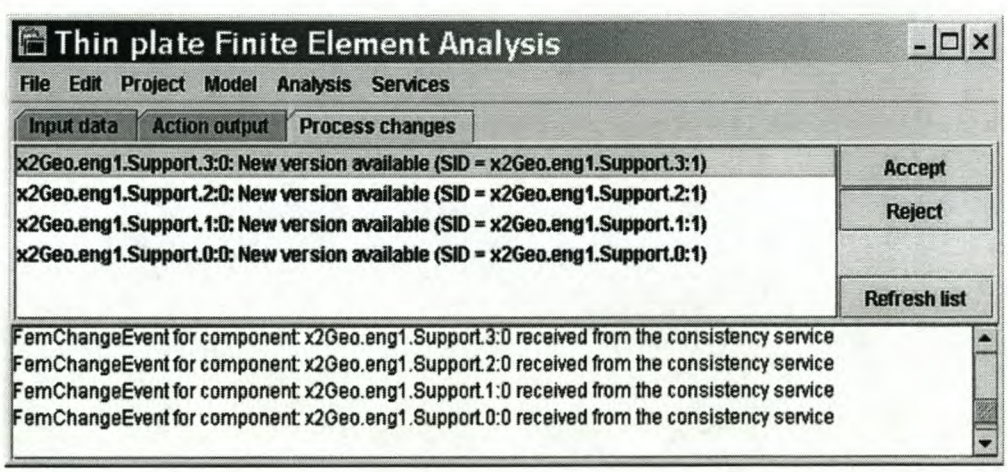
4. Since all the geometric components, including the Supports, were checked out in consistent mode, the changes above are forwarded by the workplace consistency service to the central consistency service. In turn, the central consistency service forwards a message about each change to all members of the analysis group (e.g. eng1) that have checked-out the same parameters in consistent mode, and have started their workplace consistency services. Furthermore, if a member checks out a parameter in consistent mode at a later point in time, and some changes have been made to the parameter before that time, he will receive all the accrued messages as well. The graphic below shows the current state of the Supports in the workspace of eng1, and the messages from the workplace consistency service about incoming changes.



5. The member receiving the messages can react to them at a convenient time by executing Services /> Consistency /> Process changes. On the "Process changes" tabbed pane information about the changes are displayed. The messages in the graphic below indicate changes of support codes from SIMPLE (0) to FIXED (1). The member can select any set of the changes and either Accept or Reject them. If a change is Accepted, the corresponding parameter in the local model is updated.



6. Changes that are made to parameters can only be saved in the PAPD if a new version of the parameter is created. Changes can be accumulated until a member feels that the state of a model reflects a valid configuration before new versions are created. Whenever a new version of a consistent parameter is created, the new version is made available through the consistency service to other consistent users of the parameter. Note the new version numbers of the Support components in the graphic below. The user can again choose to Accept or Reject any of the modifications. If he accepts a new version, the complete newly versioned parameter supercedes its counterpart in his local address space.



7. After bringing about the necessary changes to the geometric components, and having added the loading components, the model is analysed as described in example 1. The model and its new components are then checked into the PAPD. New versions of the Support components, which were checked out in consistent mode, are checked in with the model. A record of consistent mode evolution of components are kept in the PAPD in the table tConsistentEvolution. This is useful in evaluating the validity of a model at checkout time: if some of its components have been modified in consistent mode there is a chance that the model is still valid, but the updates have to be investigated. Table tConsistentEvolution simply lists the persistent identifier, the version number of the component at consistent checkout time, and the version number at checkin time.

Table tConsistentEvolution

Change_ID	pid	checkOutVersionNumber	checkInVersionNumber
1	x2Geo.eng1.Support.0	0	1
2	x2Geo.eng1.Support.1	0	1
3	x2Geo.eng1.Support.2	0	1
4	x2Geo.eng1.Support.3	0	1

The new versions of the Support components are added to table tSupports:

Table tSupports

pid	versionNumber	versionTime	versionHistory	ownerId	taskgroupId	accessCode
x2Geo.eng1.Support.0	0	1019736809909		eng1	geoGrp	63
x2Geo.eng1.Support.0	1	1019892903411	0	eng2	l&aGrp	63
x2Geo.eng1.Support.1	0	1019736809909		eng1	geoGrp	63
x2Geo.eng1.Support.1	1	1019892903401	0	eng2	l&aGrp	63
...						

nodeId	supportCode	status1	status2	status3	value1	value2	value3
x2Geo.eng1.Node.0	0	True	False	False	0.00E+00	0.00E+00	0.00E+00
x2Geo.eng1.Node.0	1	True	True	True	0.00E+00	0.00E+00	0.00E+00
x2Geo.eng1.Node.1	0	True	False	False	0.00E+00	0.00E+00	0.00E+00
x2Geo.eng1.Node.1	1	True	True	True	0.00E+00	0.00E+00	0.00E+00
...							

10.2 Evaluation

A framework for distributed structural analysis has been proposed and implemented in prototype form as described in chapter 9. In this section an evaluation of the proposed solution is based on measuring certain performance characteristics of the prototype implementation with the principal aim of estimating the overheads involved in working in a distributed collaboratory as opposed to performing a standard structural analysis on a local workstation. Since the proposals addressed only technical aspects of distributed collaborative analysis, the evaluation is restricted to these aspects.

Local actions: A large number of applications treat structural analysis in a local environment, and these applications are widely used. Their usage requires an effort which varies, depending on the nature of the problem under consideration and the tools provided by the application. A substantial part of the work that is done in a distributed collaboratory is not constantly influenced by or dependent on the collaboratory services, and the effort it requires is comparable to that of local environment applications. In total, however, working in a distributed collaboratory requires additional effort.

Collaborative actions: Members of an analysis group will experience the impact of working in a distributed collaboratory by having to deal with the following:

- Checking parameters into the project analysis parameter database (PAPD).
- Checking parameters out of the PAPD.
- Versioning parameters that are created or modified.
- Assigning persistent identifiers to parameters that are created.
- Reacting to changes propagated through the consistency service.

The additional operations required by the above actions and the volume of data associated with each action are overheads which require execution time on both the project server and the local workstation, as well as time for transferring the associated data over the network. The resulting latencies which arise due to working in a distributed collaboratory are experienced negatively by users and are the focus of this evaluation.

Latency: The latency associated with a collaborative action in general has three components:

- Time required to process the action on the local workstation.
- Time required to transfer the data associated with the action over the network between the workstation and the project server.
- Time required to process the action on the project server.

Since processing times depend on the computational capacities of the computers that are used, and the network transfer time on the properties of the network that is used, measured latencies will vary from configuration to configuration. However, the number of operations required for an action, and the volume of data it needs to have transferred over the network

are constants for a given implementation. As a result operation counts and data volumes, being configuration independent indicators of latency, are used as the primary indicators of the overhead cost of distributed work.

10.2.1 Procedure

Implicit operation count: An explicit determination of the number of operations required for a specific action would require an analysis of the Java Runtime Environment and its interaction with the operating system. Such an analysis would be time consuming and would not lead to additional insight. To avoid this the operations required to execute a specific action are counted implicitly. The implicit operation count is the number of unit operations required for completion of the action, and is determined by dividing the computation time required for the action by the computation time required for a chosen unit operation.

Unit operation: A very simple operation, namely the local assignment `a=b`, was chosen as the unit operation. The unit operation's computation time was determined by measuring its cumulative time over a large number of repetitions[1] and dividing by the number of repetitions. The number of repetitions were increased until a constant result for the unit operation time was reached. This is necessary because the time resolution of the computers on which the tests were performed is restricted to milliseconds. During these measurements, care was taken to avoid interferences, e.g. Java's automatic garbage collection. Furthermore, to account for the load effect of the running prototype application during testing, the unit operation time was computed just prior to the determination of each action's operation count.

Operation count for an action: Determining the implicit operation count of an action requires measuring the time required for completion of the action. To account for the millisecond time resolution of the test computers, the time required by short duration actions were taken over a large number or repetitions and divided by the number of repetitions. It was not practical to increase the number of repetions until a stable time requirement could be attained. Instead, each experiment was repeated at least five times using a number of repetitions sufficiently high to yield a coefficient of variation of less than one percent amongst the experimental results.

[1] The looptime is subtracted.

10.2.2 Results

Local, network and server operations: Each of the collaborative actions listed in section 10.3 trigger operations which take place on the local workstation, transfer a certain volume of data over the network, and cause operations to take place on the project server. Implicit operation counts and network data volumes associated with these actions are dealt with in the following sections. By considering the local workstation, the network and the server operations seperately, the overheads associated with working in a distributed collaboratory become clearer and can be compared more rationally to the alternative of working only in a local environment.

Particular results: Certain analysis parameters, e.g. the finite element `Model` and the primal and dual vectors of the `Equation`, do not have a fixed size. Consequently the number of operations they require for a specific action cannot be counted, since it directly depends on the size of the parameter under consideration. For such cases particular results, pertaining to specific model sizes as well as a given technology configuration, are listed. Particular results were generated by considering a square plate meshed with an even number of subdivisions in both directions. Models with 32, 64 and 96 subdivisions per side were used. The plate is loaded over its entire surface with an area load, as well as node loads at each node. The plate is simply supported along its four edges. The local workstation was a 900MHz Pentium III Windows 2000 machine, the project server similar, except with a processor speed of 750MHz. The network was a 10 Megabits per second local area network.

10.2.2.1 Checking parameters into the PAPD

Local, network and server operations: When checking parameters into the PAPD, local operations are performed in order to

- pack each parameter into the streamed format (byte sequence) suitable for network transfer to the project server. The number of bytes representing the parameter in streamed format is the payload which needs to be transported over the network.

Network operations are required to transfer the streamed format of the parameter to the project server.

On the project server two sets of operations are required, namely

- operations involved in unpacking the streamed format of the parameter, and
- database-operations required to store the parameter in the PAPD.

The average number of operations and the payload data volumes of the prototype implementation are listed in table 10-1.

Parameter type	Local operations	Network payload (bytes)	Server operations	
	Pack into streamed format		Unpack streamed format	Database storage
Node	3436	160	4026	280732
Local	3729	197	4587	255576
Material	3537	151	4543	252003
Element	5489	233	7932	279954
NodeLoad	3983	184	4480	263420
LineLoad	5219	233	6353	288415
AreaLoad	3981	186	4490	265124
Support	4049	197	5197	285154

Table 10-1: Parameter check-in operations and data volumes.

Distributed environment overhead: The operation counts for database storage listed in table 10-1 are the result of the database technology chosen for the PAPD. If the same database technology and implementation were used to store parameters on the local workstation, the number of database storage operations required will be the same as the corresponding values listed in table 10-1. The distributed environment overhead is the sum of the packing and unpacking operations and the network operations required to transport the streamed format. Excluding network transport, the overheads caused by combined packing and unpacking of parameters to and from the streamed format varies between 2,7%[1] and 4,8%[2] of the number of storage operations. The perception that these distribution overheads are low has to be tempered by two facts:

1. The operation counts for storage in a relational database, chosen specifically for the distributed environment, are probably significantly higher than what might be required for a more effective local storage technology.
2. Although the streamed package sizes are small, the cost of network transfer could be high, especially in the general case where the Internet has to be used.

Buffering: From the point of view of a member of the analysis group, operations that are performed by the network system and those which take place on the project server do not inconvenience him if he does not have to wait for their completion. This situation could occur if parameters are checked into the PAPD using a buffering mechanism: as soon as the parameters have been flushed to the network stream the network system is responsible for transferring them to the project server and the project server is responsible for further operations to unpack and store them, while the local workstation resources are available for other actions. In this case the overhead associated with distributed work is the difference between

[1] for `Node` parameters
[2] for `Element` parameters

the number of operations required to flush the package containing the streamed format of the parameter to the network stream and the number required to flush the package to a local storage alternative. In order to cast some light on the operational cost of flushing a packaged parameter to a network stream, a 233 byte package[1] was flushed to three different destinations, namely an internal buffer which is accessed directly, a local disk file which is representative of a local storage alternative, and a buffer on the local computer which is reached via a socket connection. The results are listed in table 10-2.

Flushing 233 bytes to:	Operation count
Internal buffer	83
Disk file up to size 2,3 MB	182
Disk file up to size 23 MB	663
Disk file up to size 46 MB	752
Buffer via socket	1628

Table 10-2: Flushing a packaged parameter

As expected, flushing the byte package to an internal buffer is cheap. The cost of flushing to a local disk file increases as the file output stream grows, while transferring the package through a socket connection is relatively expensive. The results indicate that even if a buffering mechanism is implemented to screen the client off from the network and server latencies, there is still a distributed storage operational overhead which, compared to a local storage alternative, will be in excess of a factor two.[2] The positive effect of buffering on the responsiveness of the application, however, is important.

Particular results: Particular results were generated for the configuration described in section 10.3.2. The times required to check the different models, including their primal and dual solution vectors into the PAPD were measured. Measured parameter check-in times, in seconds, are listed in table 10-3. The results indicate total check-in times which, in terms of interactive work on a computer, are unacceptably long. However, the bulk of the total time was spent on the server. This correlates with the high number of database operations required to check a parameter into the PAPD. It also indicates that a considerable effort should go into performance optimization of the PAPD server software, and that the project server has to be a high performance computer. The network time is comparatively low, as can be expected with the local area network that was used. In general, however, a wide area network has to be used, in which case network latency will become more prominent.

[1] The largest streamed format of a parameter is 233 bytes. Furthermore, the experiments returned the same result for buffer sizes in the range of the different parameter sizes, i.e. 151 through 233 bytes.
[2] i.e.1628/752

Parame-ter type	Square plate 32x32				Square plate 64x64				Square plate 96x96			
	N	T [s]	S [s]	Nw [s]	N	T [s]	S [s]	Nw [s]	N	T [s]	S [s]	Nw [s]
Model	1	1	1	0	1	3	3	0	1	7	7	0
Equation	1	0	0	0	1	1	1	0	1	2	2	0
Node	1089	7	6	1	4225	29	26	3	9409	73	57	15
Material	1	0	0	0	1	0	0	0	1	0	0	0
Element	2048	14	12	2	8192	61	55	7	18432	156	122	33
NodeLoad	1023	6	5	1	4095	27	24	4	9215	66	52	14
AreaLoad	2048	13	11	2	8192	58	52	7	18432	149	116	33
Support	128	1	1	0	256	2	2	0	384	3	2	1
Total	6339	42	37	5	24963	182	161	20	55875	456	359	96

Table 10-3: Parameter check-in times (particular configuration)

N : Number of parameters T : Total check-in time

S : Time spent on server operations Nw : Time spent on network operations

Values are rounded. Zero indicates a time of less than half a second.

10.2.2.2 Checking parameters out of the PAPD

Local, network and server operations: When parameters are checked out of the PAPD, two sets of local operations are performed by the PAPD client, namely

- operations which pack a request for each desired parameter for transfer to the PAPD server, and
- operations which unpack the streamed format of the parameter and instantiate it when it arrives.

Network operations are required to transport the request to the server and the reply from the server to the local workstation. The reply is the streamed format of the parameter.

Three sets of operations are performed by the PAPD server, namely

- operations for unpacking and analysing the client's request,
- database operations required to retrieve the parameter's data from the PAPD, and
- operations for packing it in the streamed format for network transport to the client.

When a parameter is requested for use in consistent mode, the PAPD server also registers it with the central consistency service on the project server.

Average numbers of the operations described above, and the payload data volumes of the prototype implementation are listed in table 10-4.

Parameter type	Local operations			Network payload		Server operations			
	Pack request	Unpack streamed parameter		Request (bytes)	Reply (bytes)	Unpack request	Pack parameter		Database retrieval
		free	cons.				free	cons.	
Node	1445	4693	5281	48	151	1472	5829	10810	515025
Local	1445	5029	5480	48	188	1472	6473	11387	337508
Material	1445	5037	5516	48	142	1472	6160	11075	366575
Element	1445	8413	8787	48	224	1472	8328	12748	475228
NodeLoad	1445	5668	6375	48	175	1472	6370	11200	426961
LineLoad	1445	7029	7847	48	224	1472	7637	12742	544803
AreaLoad	1445	5680	6396	48	177	1472	6498	11270	427116
Support	1445	5962	6330	48	188	1472	6786	11329	535398

Table 10-4: Parameter check-out operations and data volumes.

free = free mode check-out cons. = consistent mode check-out

Distributed environment overhead: The operation counts for database retrieval listed in table 10-4 are the result of the database technology chosen for the PAPD. If the same database technology and implementation were used to store and retrieve parameters on the local workstation, the number of database retrieval operations will be the same as the corresponding value listed in table 10-4. The distributed environment overhead is the sum of the

respective packing and unpacking operations and the network operations required to transport the request and the streamed format of the returned parameter. Excluding network transport, the overheads caused by the packing and unpacking operations varies between 2,6%[1] and 5,9%[2] of the number of retrieval operations. Similar to the case of parameter check-in described in section 10.3.2.1 above, the perception that these distribution overheads are low has to be considered while keeping the following in mind:

1. The operation counts for retrieval from the relational database, chosen specifically for the distributed environment, are probably significantly higher than what might be required if a more effective local storage and retrieval technology were used.
2. Although the streamed package sizes are small, the cost of network transfer could be high, especially in the general case where the Internet has to be used.

Buffering: As described in section 10.3.2.1 above, a buffering mechanism may be considered to screen the client off from the latencies caused by server and network operations. During parameter check-out, such a mechanism is not of much help if the client needs the requested parameters straight away and have to wait for their arrival in any event. However, it is conceivable that a client submit a request for parameters, and continue with other work while waiting for his request to be serviced. In this case the overhead associated with distributed work comprise two sets of operations:

- Flushing the client's request to the network stream: The cost of flushing a number of bytes to different destinations, including the network stream, is listed in table 10-2. [3]
- Reading the server's reply from the network stream: The operational costs of reading a number of bytes[4] from different sources are listed in table 10-5.

Reading 233 bytes from:	Operation count
Internal buffer	116
Disk file, size 2,3, 23, 46 MB	293
Buffer via socket	404

Table 10-5: Reading a packaged parameter

Reading the byte package from an internal buffer is cheap. The cost of reading from a local disk file, which represents the local storage alternative, is independent of the size of the file input stream. However, opening large files may cause Java applications to run out of main memory. Reading the package through a socket connection requires considerably less

[1] for `Node` parameters requested in free mode
[2] for `Local` parameters requested in consistent mode
[3] The results of table 10-2 remained unchanged when a 48 byte buffer, i.e. the size of the client's request, was used.
[4] The actual number of bytes used were 233, but the results were not influenced by variation of this number within the range of the analysis parameter streamed format sizes.

operations than flushing the same package through the connection[1]. The total overhead for distributed retrieval, i.e. flush request and read reply (1628+404), compared to retrieval from local disk storage (293), is considerable, namely a factor of almost seven (7). However, as in the case of parameter check-out, the benefit of buffering in terms of the responsiveness of the application is important.

Particular results: The particular configuration and models described in section 10.3.2 were used to measure parameter check-out times. Results are listed in table 10-6, and lead to similar comments and conclusions as those that were reached for the case of parameter check-in, namely:

- The total check-out times are too long to be acceptable for interactive worksessions.
- The bulk of the total time is spent on the server, which correlates with the high number of database operations required to check a parameter out of the PAPD.
- The earlier statements, namely that considerable effort should go into performance optimization of the PAPD server software, and that the project server has to be a high performance computer, are reinforced.
- The network time is comparatively low. However, this is the result of using a local area network. Network latency will become more prominent when a wide area network has to be used.

Parame-	Square plate 32x32				Square plate 64x64				Square plate 96x96			
ter type	N	T [s]	S [s]	Nw [s]	N	T [s]	S [s]	Nw [s]	N	T [s]	S [s]	Nw [s]
Model	1	12	12	0	1	52	51	0	1	120	119	0
Equation	1	10	10	0	1	46	46	0	1	109	109	0
Node	1089	10	8	2	4225	36	33	3	9409	84	73	11
Material	1	0	0	0	1	0	0	0	1	0	0	0
Element	2048	17	14	4	8192	67	61	6	18432	166	138	28
NodeLoad	1023	9	7	2	4095	34	28	6	9215	77	67	11
AreaLoad	2048	17	13	4	8192	71	59	12	18432	166	131	35
Support	128	1	1	0	256	2	2	0	384	3	3	0
Total	6339	76	65	12	24963	307	281	27	55875	724	639	85

Table 10-6: Parameter check-out times (particular configuration)

N : Number of parameters T : Total storage time

S : Time spent on server operations Nw : Time spent on network operations

Values are rounded. Zero indicates a time of less than half a second.

[1] The shorter reading time indicates a buffering mechanism between the output stream of the source socket and the input stream of the receiving socket. For transfer in the opposite direction, i.e. flushing through the socket connection, such buffering is not possible since the flushed bytes have to be written to the underlying stream, i.e. the input buffer of the receiving socket, hence the longer flushing time.

10.2.2.3 Versioning

Local, network and server operations: Updating the version of a modified parameter requires two sets of local operations, namely

- packing a request for a new version for transport to the versioning server, and
- unpacking the versioning server's reply and using it to update the version.

Network operations are required to transport the request to the versioning server and its reply to the requesting client.

Three sets of operations are performed by the versioning server, namely

- unpacking the client's request,
- processing the request, and
- packing a reply for transport to the client.

The average number of operations of the versioning actions and the payload data volumes of the prototype implementation are listed in table 10-7.

Local operations		Network payload		Server operations		
Request pack	Reply unpack	Request (bytes)	Reply (bytes)	Request unpack	Reply pack	Process request
3955	2441	46	18	3068	2683	258

Table 10-7: Versioning operations and data volumes

Distributed environment overhead: Structural analysis in local environments is traditionally performed without support for the versioning of analysis parameters. If, for the purpose of comparison, versioning is assumed to be applied in local environment analyses, the overhead associated with distributed versioning turns out to be high. As indicated in table 10-7, processing of a versioning request is cheap at 258 operations. Furthermore, a local method call with parameter and return value similar to that required for the versioning process requires only 7 operations. Compared to that, the total of 12147 operations[1] for the packing and unpacking listed in table 10-7, which still excludes the operations required for network transport, is high.

Buffering: The versioning client can be screened off from the latencies caused by server and network operations through buffering, provided the version-updating information is not required right away. However, the local packing and unpacking operations totalling 6396[2] would still be required, as well as operations totalling 2032[3] for flushing and reading a number of bytes through a socket connection. When compared to the 7 operations needed for the method call required for local versioning, the operational overhead of versioning in a

[1] 3955+2441+3068+2683
[2] 3955+2441 from table 10-7
[3] 1628+404 from tables 10-2 and 10-5

distributed environment is high, even when buffering is used. However, its impact on the performance of the local workstation, i.e. the latency it causes, will be negligible if the versioning client executes in a low-priority thread.

Particular results: Version-updating times for the models and local area network configuration described in section 10.3.2 were measured. Since the versioning mechanism is the same for all parameters, the time required is directly proportional to the number of version updates requested. Small deviations due to the local area network's load-state are visible in the results listed in table 10-8 below. As expected, the bulk of the time is spent on preparing and transferring the update request and its corresponding reply. The times listed in table 10-8 are not excessive if the fact is considered that only new models will require version updating for a large number of parameters at a time. However, in the general case where the Internet has to be used, the times may become unacceptably long and the use of a buffering mechanism is advisable.

Model	Number of parameters	Time [s]	
		Pack and Network	Total
Square plate 32x32	6339	3.61	3.71
Square plate 64x64	24963	14.64	15.02
Square plate 96x96	55875	29.05	30.17

Table 10-8: Version updating times (particular configuration)

10.2.2.4 Persistent identification

Buffering by design: In section 7.3.3 the fact that new parameters are created in interactive worksessions which demand fast response was made central to the design of the persistent identifier service (PID service). A buffering mechanism was proposed and implemented in the prototype service. Once the PID client has been configured and started, it retrieves identifiers for new parameters from the PID server and caches them in a local buffer.

Local, network and server operations: The buffering of identifiers requires two sets of local operations, namely

- packing a request for a new identifier for transport to the PID server, and
- unpacking the server's reply and storing it in the local buffer.

Network operations are required to transport the request to the PID server and its reply to the requesting client.

Three sets of operations are performed by the PID server, namely

- unpacking the client's request,
- processing the request, and
- packing a reply for transport to the client.

The average number of operations required for persistent identification, and the payload data volumes of the prototype implementation are listed in table 10-9.

Client operations		Network payload		Server operations		
Request pack	Reply unpack/buffer	Request (bytes)	Reply (bytes)	Request unpack	Reply pack	Process request
3177	662	29	32	1646	2704	1014

Table 10-9: Persistent identification operations and data volumes

Distributed environment overhead: The operational overhead due to the distributed configuration of the PID service is the sum of the packing and unpacking operations listed in table 10-9, which total 8188 [1], plus network operations for the transfer of approximately[2] 60 bytes. This implies an overhead factor of more than eight (8) compared to the operational cost of 1014 operations needed to process the request locally. However, the buffering mechanism that was implemented practically eliminates the negative impact of distribution on latency. Reading a number of bytes from an internal buffer requires only 116 operations as listed in table 10-5. As a result user access to identifiers is extremely fast. Furthermore, the PID client is a low priority thread and its impact on the response of the local workstation

[1] 3177+662+1646+2704

[2] The data volume of both the request and reply depend on the length of certain identifier strings, e.g. the task identifier.

is negligible. Consequently, in terms of latency the distributed nature of this service is transparent to users.

Particular result: The PID client attempts to maintain a cache of identifiers for each of the different types of analysis parameters, and the size of each cache can be set by the project coordinator. Using the configuration described in section 10.3.2, the PID client retrieved identifiers for 2 Models, 2 Equations, 1000 Nodes and Elements, 500 NodeLoads and AreaLoads, 250 Locals, LineLoads and Supports and 10 Materials, i.e. a total number of 3764 identifiers, in two (2) seconds. For cases where network communication is slow, larger cache sizes will be required to prevent caches from running empty during modeling sessions in which a large number of new parameters are created over a short period of time.

10.2.2.5 Propagation of changes to consistent parameters

Local, network and server operations: Fundamental changes to consistent parameters are propagated by the consistency service of the collaboratory. The workplace consistency service requires three sets of operations in the execution of its tasks, namely

- detecting changes to local consistent parameters and packing a corresponding change notification[1] for transport to the central consistency service,
- unpacking and buffering change notifications received from the central service, and
- executing an update of a parameter if the user chooses to implement an incoming change.

Network operations are required to transport change notifications between the workstation and the central consistency service.

The central consistency service requires two sets of operations, namely

- unpacking change notifications received from the workplaces and buffering them for forwarding to other workplaces, and
- processing the list of consistent parameters and packing change notifications for transport to each workplace as and if required.

The average number of operations integral to the sets described above, and the network payload data volume of the prototype implementation are listed in table 10-10 [2].

Workplace operations			Network payload	Server operations	
Detect and pack notification	Unpack and buffer incoming notifications	Execute update	Change event (bytes)	Unpack and buffer incoming notifications	Process and pack outgoing notifications
9544	9192	20102	112	9853	135371

Table 10-10: Consistency operations and data volume

Overhead:. The computational requirements of the consistency service listed in table 10-10 have an impact on the performance of the member workstations and the project server, and it burdens members with the task of attending to change notifications. These are overheads compared to the effort required for standard structural analysis in a local environment. However, this overhead cost cannot be measured in terms of an equivalent local-workstation-only alternative, since a consistency service only makes sense in a distributed environment where versioned parameters are used at different locations. Considering only the task of modifying a parameter, an average of 72 operations are required for a local `set`-method to implement the modification. Compared to this a total of 9192+20102=29294 operations are

[1] Instance of class FemChangeEvent, see section 9.5.3.1

[2]. The results vary slightly for different parameters and for the particular type of change effected to a parameter. The listed values are represent average results

required to unpack, buffer and execute an update of a parameter. Clearly, achieving a modification through the update-procedure is expensive. However, from the user's perspective latency is of greater importance.

Latency: The consistency service implements a buffering mechanism where local changes are automatically propagated and remote changes are automatically received in low-priority threads. Whenever the user decides to process received changes, he only deals with changes that are available in the local buffer. Although the updating action requires a large number of operations more than simple modification, the latency introduced by these operations is negligible compared to the time the user has to devote to deciding whether to accept or reject an incoming modification.

Particular results: The speed with which notifications of fundamental changes are propagated to the members of the analysis group is on the whole not critical since much of the collaborative work will take place asynchronously. Using the computer and network configuration described in section 10.3.2, changes propagated from one workplace to another in less than two (2) seconds. If a much slower network is used the propagation time might increase to the order of a minute. Such a time lapse is neglible, even in the case of members working synchronously.

11 Conclusion

The research described in this dissertation was aimed at the structure of information and communication in a structural analysis collaboratory whose workplaces are nodes in a communication network. Towards this aim the essential information requirements and functionalities of an application that can support the fundamental tasks of structural analysis were examined, initially within the context of analysis in a local environment and subsequently within the context of the distributed collaboratory. Linear elastic analysis of thin plates was considered in detail, but the results can be readily extended to accommodate other types of analysis and structural components. Information structures and services for the management of information in local and distributed environments were developed and implemented in pilot form.

Persistent identification: The structuring of applications around persistently identified objects proved to be crucial. Specifically, the structure of the analysis model, which is laid down in relations on its component parameters, is described in terms of the persistent identifiers of parameters. The reference-connections required for processing are activated on demand by enabling each parameter to establish its connections using the specified persistent identifiers and an object map maintained by the application. This technique was extended to provide for the fact that the project analysis parameters are versioned by introducing selection identifiers. The selection identifiers are used when dealing with distribution aspects of the application, while the persistent identifiers are used for local processing.

Crucial advantages: The effect of structuring the distributed analysis application around the use of persistent identifiers is comprehensively beneficial. In particular, two advantages come to the fore:

- Exchange of analysis parameters amongst members of the analysis group is technically straightforward since reference-connections are easily established after a parameter has been transferred and instantiated in the local workspace. Furthermore, a parameter can be referred to during modeling even though it has not yet been instantiated in the local workspace.
- The structure of the project database is simplified considerably. Relationships amongst parameters are described using persistent identifiers (or selection identifiers in the case of Models), and these are stored as fundamental attributes of each parameter. Reference-connections are taken care of by the application. If a referenced object is not available in the workspace, the problem is easily identified and corrected.

Information management and collaboration utilities: The crucial advantages of using persistent identification to structure information and communication in the collaboratory made solutions possible in which the following could be achieved:

- ***Standalone capability:*** For non-collaborative work, full standalone capability is provided. During collaborative worksessions, necessary analysis parameters that were created at remote locations are transferred to the local workstation whenever network communication with the project server is possible. Once the transfer has been completed, the parameters are assimilated and used in a local model in the absence of further network communication.
- ***Consistency:*** Strict consistency of analysis parameters is not enforced, and fundamental attribute values in the project database are never modified. Instead, new versions of parameters are simply added to the database. Whenever a member checks out a parameter for use in consistent mode, he is informed if a later consistent version is available. During the period that parameters are checked out in consistent mode, the consistency service informs all members that are using them of changes made to the parameters, and provides them with the opportunity to either accept or reject the changes.
- ***Flexibility, media-suitability and navigability:*** The extent to which it is possible to purposefully select a sub-set of data units contained in an information base eventually determines whether tools can be developed that support users in finding, and flexibly selecting essential information for retrieval. The simple structure of the project database facilitates the development of such tools and offers the potential to increase their flexibility through support of user programming at the interface level.
- ***Information-base updating time:*** The contents of the project database is only altered through an explicit action on the part of a member of the analysis group, namely the action of checking parameters into the database, which may take place at any time chosen by the member. New versions of modified parameters have to be created before attempting to check such parameters into the project database, and the updating of versions is also takes place under the control of the users.
- ***Information-base updating area:*** The members of the analysis group have complete freedom regarding the choice of parameters that they check into the project database. A member may, for example, choose to check only one or more `Node` parameters into the database.
- ***History of decisionmaking:*** The versioning history of all parameters, and the consistent version evolution of parameters that are used in consistent mode are available in the project database. The user name of the member who created each parameter, and the time of creation is recorded as well. Reasons for modifications are not recorded.
- ***Reliability and safety:*** The integrity of project database is not threatened by system or network failures since parameters are checked into and out of the database in transactions whose success can be monitored. Editing and processing of parameters take place

on workstations at the workplaces. As a result the same degree of reliability and safety offered by standard local applications can be achieved.

- ***Maintaining a feeling of collaboration:*** By logging in to work on a project, choosing a specific task to work on, checking parameters into and out of the project database, updating versions and dealing with messages from the consistency service regarding changes at remote locations members of the analysis group are constantly being made aware of their participation in the collaboratory.

The achievements listed above were identified at the commencement of the research as criteria according to which the success of collaboration-supporting utilities and information management in a collaboratory should be measured. Furthermore, the evaluation of operational aspects of the pilot application indicated areas of satisfactory performance, and areas where performance optimization of implementations is required. Ways of limiting the latency associated with distributed work were discussed, and are considered to be practicable. On this basis the conclusion is reached that the aims of the research have been attained.

Recommendations: The extent of the research effort aimed at the design of collaboration-supporting technologies for the building industry was described in the opening statements of this dissertation. A large number of known problems are being addressed, and there is no shortage of pressing topics that need to be researched.

The importance of

- persistently identifying objects,
- using persistent identifiers to describe relations between objects, and
- structuring applications around the use of persistent identifiers to establish references at runtime

was the central theme of this dissertation and these have to be assimilated in new developments.

Furthermore, the importance of two areas of research are singled out:

- ***Standardization:*** An essential feature of the distributed analysis model described in this dissertation is the existence of a standardized product model. The importance of continued research into the standardization of information content in the civil engineering industry should not be underestimated.
- ***Consistency:*** The consistency service described in the dissertation is considered to have practical value within the context of structural analysis as an isolated task. For distributed engineering in general, however, the reliability and efficiency of procedures for updating and securing the consistency of the distributed information base will be vital.

References:

Abaqus Finite Element Analysis Products, Hibbitt Karlsson & Sorensen Inc., http://www.abaqus.com/

Abaqus Explicit: http://www.abaqus.com/products/p_abexplicit.html

Adina R&D Inc., Finite Element System for Structures, Heat Transfer and CFD, http://www.adina.com

Al-Qawasmi, J., Clayton,M.J., *Media Usage: Observations from an Experimental Study of Computer-Mediated Collaborative Design*. In Fruchter, R, Ed., Proc. 8th Int. Conf. on Computing In Civil and Building Engineering, p.860-867, Stanford, California, Aug. 14-16 2000, ASCE.

Bentley,R., Rodden,T., Sawyer,P., Sommerville,I., *Architectural support for cooperative multiuser interfaces.*, IEEE Comp., May 1994.

Boger, M., Java in verteilten Systemen., dpunkt.verlag, Heidelberg, 1999, ISBN 3-932588-32-0

Borghoff, U.M., Schlichter, J.H., Computer-Supported Cooperative Work, Introduction to Distributed Applications, Springer-Verlag, 2000, ISBN 3-540-66984-1.

Bretschneider, D., Modellierung rechnerunterstützter, kooperativer Arbeit in der Tragwerksplanung, Fortschritt-Berichte VDI Reihe 4 Nr. 151, VDI Verlag, Düsseldorf, 1998, ISBN 3-18-315104-9

Brown, A. et al, *The Architecture and Implementation of a Distributed Computer Integrated Construction Environment*, CIB Workshop: Construction on the Information Highway, Bled, Slovenia, 1996

Brown, A. et al, *Promoting Computer Integrated Construction through the use of Distribution Technology* , Int. J of Construction Information Technology, **1**(1), pp.1-16, 1996

Chan, F.L., Spiller, M.D., Newton, A.R., *Weld: An environment for Web-based electronic design*, IEEE/ACM International Conference on CAD, San Fransisco, CA, 1998.

CIFE, Center for Integrated Facility Engineering, http://www.stanford.edu/group/CIFE/

Clough, R., Penzien, J., Dynamics of Structures, McGraw-Hill, 1975, ISBN: 0-07-011392-0.

Cook, R.D., Concepts and Applications of Finite Element Analysis 2nd ed., Wiley, New York, 1981, ISBN 0-471-03050-3

CORBA: Common Object Request Broker Architecture, http://www.corba.org/standards.htm

de Borst, R., Meyer, C., *Numerishce Probleme bei nichtlinearem Tragwerksverhalten.* In Mehlhorn, G., Hrsg., Rechnerorentierte Baumechanik, p. 427-488, 1996, Ernst & Sohn, Berlin, ISBN 3-433-01572-4.

DFG SPP 1103: DFG-Schwerpunktprogramm 1103: Vernetzt-Kooperative Planungsprozesse im Konstruktiven Ingenieurbau, http://www.iib.bauing.tu-darmstadt.de/dfg-spp1103

Eastman, C., ISFAA-97 Electronic Forum: *Information Sources for AEC Applications*, WG3/T12 AEC/Building & Construction group, http://www.wt.com.au/~ausstep/aec-libraries/w1-reference/eastman.html, 1997

Farley, J., Java Distributed Computing, O'Reilly& Associates, 1998, ISBN: 1-56592-206-9

Fischer, M. and Froese, T., *Examples and Characteristics of Shared Project Models*, J. of Computing in Civil Engineering, **10**(3), pp.174-182, 1996

Froese, T., *Models of Construction Process Information*, J. of Computing in Civil Engineering, **10**(3), pp.183-193, 1996

Fruchter, R. et al, *To See or Not to See: The Role of Visibility and Awareness in Videoconference-Mediated Teamwork.* In Fruchter, R, Ed., Proc. 8th Int. Conf. on Computing In Civil and Building Engineering, p.852-859, Stanford, California, Aug. 14-16 2000, ASCE.

Gamma, E. et al, Design patterns: elements of reusable object-oriented software, Reading, Mass. : Addison-Wesley, c1995. 395 p.

Gisolfi, D., Web services architect Part 3: Is Web services the reincarnation of CORBA?, http://www-106.ibm.com/developerworks/library/ws-arc3/index.html

Goodman, N., *An Object-Oriented DBMS War Story: Developing a Genome Mapping Database in C++*, In: Kim, W., editor: Modern database systems: the object model, interoperability and beyond, ACM Press, 1995

Hansen, D.M., Adams, D.R., and Gracio, D.K., *In the Trenches with ObjectStore*, Theory and Practice of Object Systems, Vol. 5(4), 201-207, 1999.

Harms, J., *Das Netz is das System.*, In Technologie-Vermittlungs-Agentur Berlin e.V.(Hrsg.)., Synergie durch Kommunikation: Netzwerksysteme, Netzwerkplanung, Multimedia., p. 21-31, Berlin, 1993, VISTAS, ISBN 3-89158-102-5

Herrera, I., *Trefftz Method.* In Brebbia, C., Editor, Topics in Boundary Element Research, p. 225-253, 1984, Springer-Verlag, Berlin, ISBN 3-540-13097-7.

Huebner, K.H., Thornton, E.A., The Finite Element Method for Engineers, John Wiley & Sons, New York, 1982, ISBN 0-471-09159-6

IAI: International Alliance for Interoperability, http://iaiweb.lbl.gov/

Illieva, D., Pahl, P.J., *Eine Datenbahn für das Bauen*, Abschlussbericht an die Technologiestiftung Innovationszentrum Berlin, Institut für Bauingenieurwesen, TU Berlin, Juli 1997.

ISO 10303: The STEP project, http://www.nist.gov/sc4/www/stepdocs.htm

Java: The source for Java technology, http://java.sun.com

Kwaw, Edward K.A., and Gorny, Peter, *Reality in Virtual Construction Using Virtual CAD (VCAD)*. In Fruchter, R, Ed., Proc. 8th Int. Conf. on Computing In Civil and Building Engineering, p.317-324, Stanford, California, Aug. 14-16 2000, ASCE.

Kensek, K. et al., *Augmented Reality: An Application for Architecture*. In Fruchter, R, Ed., Proc. 8th Int. Conf. on Computing In Civil and Building Engineering, p.317-324, Stanford, California, Aug. 14-16 2000, ASCE.

Knuth, D.E., The Art of Computer Programming, Volume 3: Sorting and Searching, 2nd edition, Addison-Wesley, 1998, ISBN: 0-201-89685-0.

Li, Liwu, Java Data Structures and Programming, ISBN 3-540-63763-X, Springer-Verlag, 1998

Lindholm, T. & Yellin, F., The Java Virtual Machine Specification, 2nd edition, ISBN 0-201-43294-3, Addison-Wesley

Mahmoud, Q.H., Distributed programming with Java, ISBN 1-884777-65-1, Manning Publications Co., 2000

mb-Software, Software für die Bereiche Arichitektur, Ingenieurbau und Betonfertigteilwerke, http://www.mb-software.de

Meskouris, K., Structural dynamics: models, methods, examples, Ernst, Berlin, 2000, ISBN: 3-433-01327-6

Minoli, D., Internet & Intranet Engineering: Technologies, Protocols, and Applications., Mcgraw-Hill, New York, 1997, ISBN 0-07-042977-4

Molkenthin, F., Holz, K.-Peter, *Working Process in a Virtual Institute*, Conference Hydroinformatic '98, Kopenhagen, 1998

Morgan, M., Java 2 for Professional Developers, Sams Publishing, 1999, ISBN: 0-672-31697-8

Nastran, MSC Software, http://www.mscsoftware.com

Nemetschek AEC Software, http://www.nemetschek.com/

Netmeeting, http://www.microsoft.com/windows/Netmeeting/features/default.asp

Oberquelle, H., *Kooperative Arbeit und menschengerechte Groupware als Herausforderung für die Software-Ergonomie*, in: Kooperative Arbeit und Computerunterstützung: Stand und Perspektiven, Verlag für Angewandte Psychologie, Göttingen, 1991, ISBN: 3-87844-018-9.

OMG: Object Management Group, http://www.omg.org/

Pahl, P.J., *Der Ingenieur und das Denken*, Festvortrag anlässlich der Ehrenpromotion, Bauhaus-Universität Weimar, 13. Oktober 1999

Pahl, P.J., Plates: Finite Element theory and Computer implementation, Skript: Theoretische Methoden der Bau- und Verkehrstechnik, TU Berlin, 2000

Pahl, P.J., Beucke, K, *Neuere Konzepte des CAD im Bauwesen: Stand und Entwicklungen*, Keynote Vortrag zum IKM2000, Bauhaus-Universität Weimar, 22-24 Juni 2000.

PBL Lab, http://pbl.stanford.edu/

Rausch, P., *Informatik für Ingenieure.*, Vieweg & Sohn, Braunschweig, 1991, ISBN 3-528-04117-X

Rezgui, Y. et al, *An Integrated Framework for Evolving Construction Models*, Int. J. of Construction Information Technology, **4**(1), pp.47-60, 1996

Rezgui, Y. et al, *An information management model for concurrent construction engineering*, Automation in Construction, **5**, pp.343-355, 1997

Rezgui, Y., Cooper, G. and Brandon, P., *Information Management in a Collaborative Multiactor Environment: The COMMIT Approach*, J. of Computing in Civil Engineering, **12**(3), pp.136-144, 1998

Riks, E., *An incremental approach to the solution of snapping and buckling problems*, Int. J. Solids & Structures, Vol. 15, p.529-551, 1979

Roman, Steven, Access Database Design and Programming, 2nd edition, ISBN 1-56592-629-9, O'Reilly, July 1999

Rosenberg, D., *Online Information Environments: Exploring Collaborative and Coordinating Technologies.* In Fruchter, R, Ed., Proc. 8th Int. Conf. on Computing In Civil and Building Engineering, p.868-873, Stanford, California, Aug. 14-16 2000, ASCE.

Rosenman, M.A., Gero, J.S., *Modelling multiple views of design objects in a collaborative CAD environment*, Computer Aided Design, Vol 28, No 3, pp 193-205, 1996.

Rüppel, U., *Vernetzt-kooperative Planungsprozesse im Konstruktiven Ingenieurbau*, Antrag auf Einrichtung eines DFG-Schwerpunktprogrammes, Institut für Numerische Methoden und Informatik im Bauwesen, Technische Universität Darmstadt, 1999.

Schneider, U., Beucke, K., *Integration standard based on a communication standard*, IABSE Symposium "Structures for the Future – The search for quality", Rio de Janeiro, 1999.

Schutte, G., Eine Erweiterung des Prinzips der virtuellen Verschiebungen zur Ermittlung von Spannungen, Dissertation, Technische Universität Berlin, Fachbereich 9: Bauingenieurwesen und Angewandte Geowissenschaften, 2000.

Shames, I.H., Dym, C.L., *Energy and Finite Element Methods in Structural Mechanics*, Hemisphere, Washington, 1985, ISBN 0-07-056392-6

Softwired Inc., Pure Java messaging solutions for electronic business, http://www.softwired-inc.com

STEP-IAI, *Guidelines for Collaboration between STEP and IAI*, ISO TC184/SC4 and the IAI, GS/12/12/97, 1997

Turk, Z.,*Communication Workflow Approach to Computer Integrated Construction (CIC)*. In Fruchter, R, Ed., Proc. 8th Int. Conf. on Computing In Civil and Building Engineering, p.1094-1101, Stanford, California, Aug. 14-16 2000, ASCE.

WebEx, http://www.webex.com

WebServices: Web Services, http://www-106.ibm.com/developerworks/webservices

WELD, The WELD project, http://www-cad.eecs.berkeley.edu/weld.

White,S., Fisher,M., Catell,R., Hamilton,G., Hapner,M., JDBC™ API Tutorial and Reference: Universal Data Access for the Java™ 2 Platform, 2/e, ISBN 0-201-43328-1, Addison Wesley Longman, June 1999.