Data Modelling of Industrial Steel Structures

by

Daniel Rudolph Oosthuizen

Thesis presented in partial fulfilment of the requirements for the degree of
Master of Engineering (Civil)
at the University of Stellenbosch

SUPERVISOR:
Dr. G.C. van Rooyen

STELLENBOSCH
December 2003

DECLARATION

I, the undersigned, declare that the work contained in this thesis is my own original work and has not been submitted in its entirety or in part for a degree at any other university.

December 2003

SYNOPSIS

AP230 of STEP is an application protocol for structural steel-framed buildings. Product data relating to steel structures is represented in a model that captures analysis, design and manufacturing views.

The information requirements described in AP230 were analysed with the purpose of identifying a subset of entities that are essential for the description of simple industrial steel frames with the view to being able to describe the structural concept, and to perform the structural analysis and design of such structures.

Having identified the essential entities, a relational database model for these entities was developed. Planning, analysis and design applications will use the database to collaboratively exchange data relating to the structure. The comprehensiveness of the database model was investigated by mapping a simple industrial frame to the database model.

Access to the database is provided by a set of classes called the database representative classes. The data-representatives are instances that have the same selection identifiers and attributes as corresponding information units in the database. The data-representatives' primary tasks are to store themselves in the database and to retrieve their state from the database.

A graphical user interface application, programmed in Java, used for the description of the structural concept with the capacity of storing the concept in the database and retrieving it again through the use of the database representative classes was also created as part of this project.

SINOPSIS

AP230 van STEP is 'n toepassingsprotokol wat staal raamwerke beskryf. Die produkdata ter beskrywing van staal strukture word saamgevat in 'n model wat analise, ontwerp en vervaardigings oogmerke in aanmerking neem.

Die informasie vereistes, soos beskryf in AP230, is geanaliseer om 'n subset van entiteite te identifiseer wat noodsaaklik is vir die beskrywing van 'n eenvoudige nywerheidsstruktuur om die strukturele konsep te beskryf en om die struktuur te analiseer en te ontwerp.

Nadat die essensiële entiteite geïdentifiseer is, is 'n relasionele databasismodel van die entiteite geskep. Beplanning, analise en ontwerptoepassings maak van die databasis gebruik om kollaboratief data oor strukture uit te ruil. Die omvattenheid van die databasis-model is ondersoek deur 'n eenvoudige nywerheidsstruktuur daarop af te beeld.

Toegang tot die databasis word verskaf deur 'n groep Java klasse wat bekend staan as die verteenwoordigende databasis klasse. Hierdie databasis-verteenwoordigers is instansies met dieselfde identifikasie eienskappe as die ooreenkomstige informasie eenhede in die databasis. Die hoofdoel van die databasis-verteenwoordigers is om hulself in die databasis te stoor asook om hul rang weer vanuit die databasis te verkry.

'n Grafiese gebruikerskoppelvlak, geprogrammeer in Java, is ontwikkel. Die koppelvlak word gebruik om die strukturele konsep te beskryf, dit te stoor na die databasis en om dit weer, met behulp van die databasis-verteenwoordigers, uit die databasis te haal.

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to those that have assisted and supported me during this research.

Dr. G.C. van Rooyen, lecturer at the Department of Civil Engineering of the University of Stellenbosch, who acted as Project Leader of the undertaken study;

Mr Martin Ruess, Technical University of Berlin, who provided me with help, expertise and guidance and for his interest in the progress of this study;

To all my friends, for their continuous support and motivation throughout the duration of this study, and

Finally, and most important of all: *TO CHRIST BE THE GLORY*

CONTENTS

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION

1.1 Overview

This study forms part of a project aimed at creating a computer-based infrastructure that supports the collaborative planning, analysis and design of an industrial steel structure. This infrastructure will be supported by a communication network. An important component of this infrastructure is a database in which product model data concerning steel structures can be stored. The database is used by the different applications to collaboratively exchange data. This is done through several interfaces (database representatives), each being specific to a certain application.

The focus of this investigation was on the definition of a product model for simple industrial steel structures, the mapping of the product model to a relational database, and the development of an interface that can serve as an example for the development of other interfaces.

Some of the applications that would make use of the database include;

- Structural concept definition: An application supporting the description of the structural frame.
- Structural analysis: Finite element analysis application
- Structural steel design: An application supporting the design of steel structures.

The overall architecture of the software system is set out in Figure 1.1.

A product model for structural steel framed buildings is described in STEP (Standard for the Exchange of Product Model Data) Application Protocol 230. This protocol describes the different structural components as well as the relationships between the components within a steel structure. Areas of the model relating to analysis describe the connectivity of elements and nodes; areas of the model relating to design describe the geometry and assembly of parts and connectors; and areas of the model relating to manufacturing describe the physical location and properties of parts and joints.

The model implemented here is a subset of STEP AP230. The design process, needed to describe a simple industrial steel structure, is investigated and the different components and relations between the components are identified.

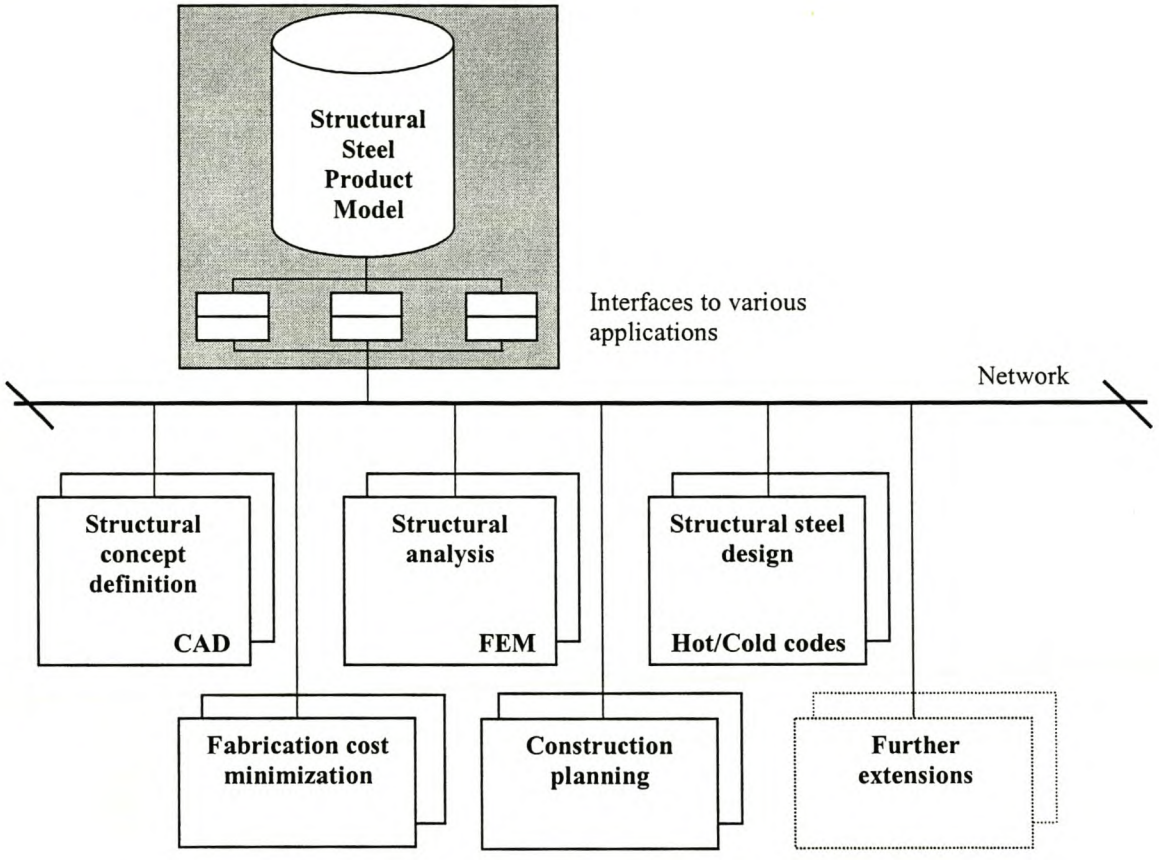


Figure 1.1: Overall Architecture

1.2 Standard for the Exchange of Product Model Data (STEP)

STEP, ISO 10303, is an international open standard for the computer-interpretable representation and exchange of product data. The objective is to provide a neutral mechanism capable of describing product data throughout the life cycle of a product, independent from any particular system. The nature of this description makes it suitable not only for file exchange but also as a basis for implementing and sharing of product databases and archiving of application objects.

Although the use of information technology (IT) is increasing rapidly within construction, no product information based on open standards yet exists for the industry. Applications used in different domains often employ very different systems, and, where data exchange does take place, it is on an ad-hoc basis. ISO 10303 addresses the growing need for an integrated approach to the use of IT within building construction.

This International Standard is organized as a series of parts, each published separately. The parts of ISO 10303 fall into one of the following series: description methods, integrated resources, application integrated constructs, application protocols, abstract test suites, implementation methods, and conformance testing. The series is described in ISO 10303-1. Application protocols provide the basis for developing implementations of ISO 10303 and abstract test suites for conformance testing of AP implementations.

The Application Protocols part of ISO 10303 specifies an application protocol (AP), namely AP230, for the representation and exchange of information relating to structural steel frames. The computer applications to which it relates are those providing analysis, member design, connection design, and detailing functions for the designers and constructors of buildings.

In AP230, product data relating to steel structures is represented in a model, which defines analysis, design, and manufacturing data capturing views.
The AP230 model is based closely on the CIMsteel Integration Standards (CIS) - which were the main deliverable of the Eureka 130 CIMsteel project. The CIMsteel project was aimed at improving the efficiency and effectiveness of the European constructional steelwork industry through the introduction of computer integrated manufacture (CIM). The CIS are ISO-aligned and were designed to offer the industry an interim solution to data exchange while providing firm foundations for the development of an ISO

standard. It was envisaged that this "twin track" approach would provide a clear migration route from CIS to ISO based technology.

AP230 defines the context, scope and information requirements for the representation and exchange of information relating to structural steel frames and specifies the integrated resources necessary for satisfying these requirements.

1.3 Product data representation and exchange - Part 230: Application protocol: Building Structural Frame: Steelwork

Application Protocol 230 of ISO 10303 specifies the information requirements relating to structural steel building frames – considered purely from the standpoint of a structural engineer.

The life cycle of a building can be decomposed into five separate stages:

(i) plan,
(ii) design,
(iii) construct,
(iv) use (including operate and maintain), and
(v) demolish.

AP230 supports the exchange of data during the first three stages of this list.

The first three stages of a building life cycle - plan, design, and construct - are completed as the result of planning, design, construction and (ongoing) management activities.

The planning and management of building quality results in the generation of project technical specifications and applicable standards - both of which govern design. In turn, project planners and managers make use of feedback from design and construction activities.

In the case of steel-framed buildings, design activities generate steelwork schedules, detailed designs, general arrangements and various types of feedback to project managers and planners. Input from designers informs and directs the fabricators and erectors of steel-framed building superstructures. Foundations are not included in this context and will not be discussed within this thesis report.

Of these information flows and activities, AP230 addresses:

- the transfer of quality-specification information into design
- design activities
- the feedback of information from designers to project planners/managers
- the transfer of information into fabrication and erection
- and the feedback of information from fabricators and erectors to project planners/managers.

Under design activities, AP230 addresses:

- structural design
- loading assessment
- structural scheme modeling
- structural analysis
- member design
- connection design
- and steelwork detailing.

The products addressed by AP230 - steelwork building frames and their components - are employed in low, medium and high rise construction in domestic, commercial and industrial applications. AP230 is applicable to a variety of structures ranging from simple, single-storey portal frame industrial buildings to multi-storey office blocks. The main structural steelwork is covered, as is secondary steelwork - such as purlins, siderails, cleats and cladding. The frames considered may be braced or unbraced. Connections can be pinned, rigid, or semi-rigid - rigid and semi-rigid being full or partial strength.

The data model underlying this AP defines frames as being fabricated from manufacturing assemblies, and describes manufacturing assemblies as being made up of parts and joint systems.

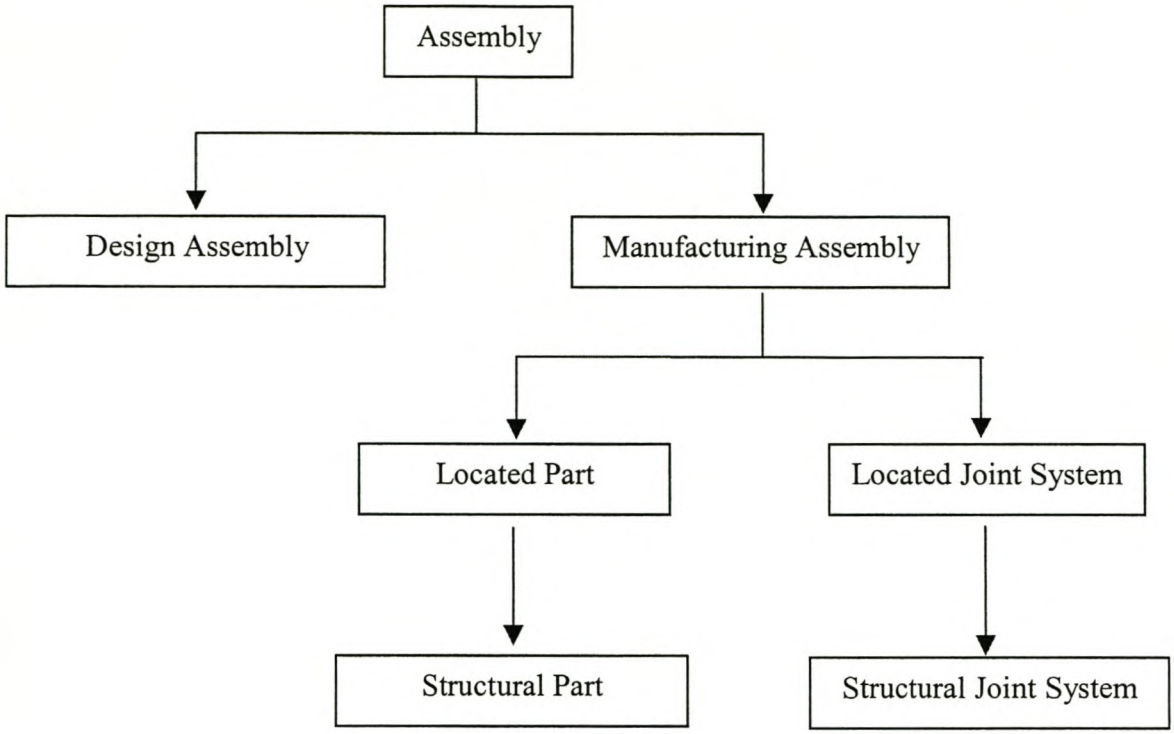


Figure 1.2: Layout of Frames

Parts may be prismatic, prismatic-derived or two-dimensional. A prismatic part is, typically, a rolled section. Prismatic-derived parts include castellated and tapered beams. Two-dimensional parts include plates and sheets.

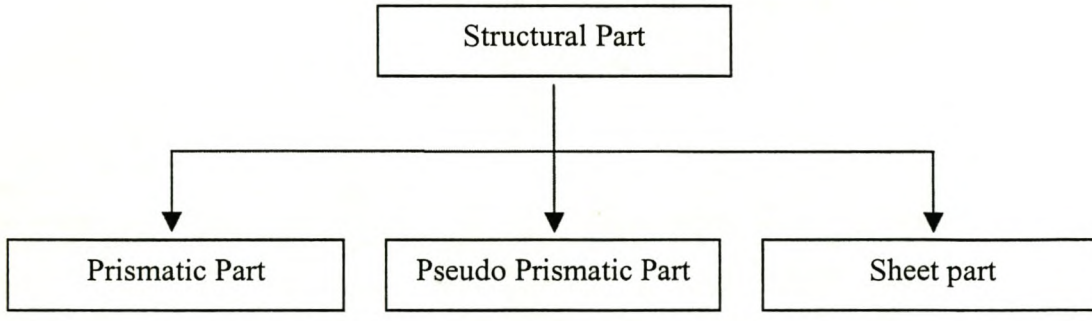


Figure 1.3: Layout of Parts

The AP includes support for rolled, welded, cast, or cold-formed parts (although only limited information is held on cast and cold-formed parts). Welded and bolted joint systems are covered, and bolted joint systems may involve ordinary and preloaded bolts.

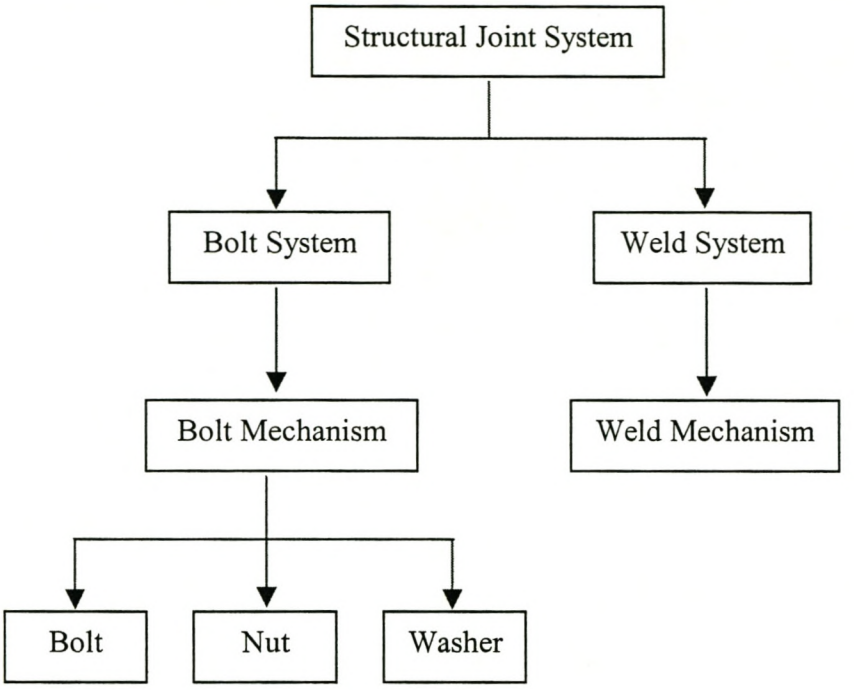


Figure 1.4: Layout of Joints

In general terms, the data supported by AP230 include:

- geometrical and geographical data
- data relating to physical and material characteristics
- data relating to structural behaviour
- data relating to unique identification
- logical grouping data
- temporal data.

CHAPTER 2

SELECTION OF STEP ENTITIES

2.1 Entities describing Structural Members

From the overview presented in section 1.3, it is clear that Application Protocol AP230 is an extended, complex document. Application protocols provide the basis for developing implementations of ISO 10303 (AP 230). Consequently AP230 was analysed in order to identify a subset of entities that is essential for the description of simple industrial steel frames.

An entity is described by means of attributes with associated data and can be defined as a static characteristic component. An entity can also be a sub-component which in combination with other sub-components make up a single component that is used to describe the super structure. An example of a component is bolt mechanism, which in itself forms an entity. A bolt mechanism is made up of several sub-components. The sub-components are a bolt, a washer and a nut. Each one of these sub-components can also be defined as an entity. The structure of STEP is shown graphically in Figure 2.1.

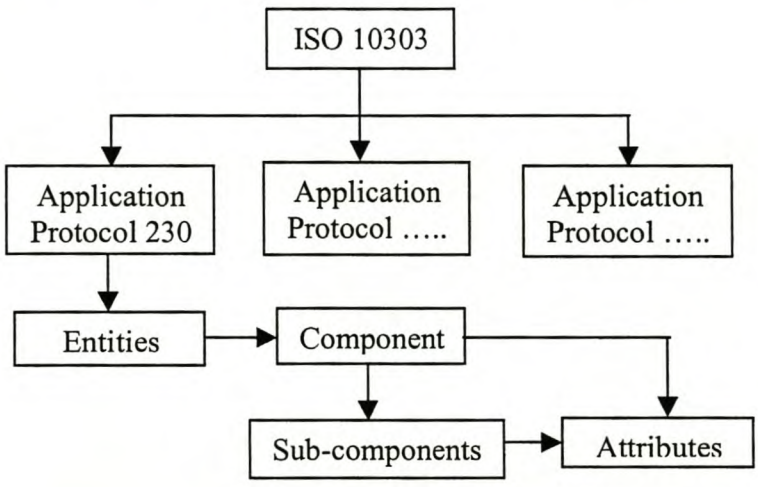


Figure 2.1: Structure of ISO standard

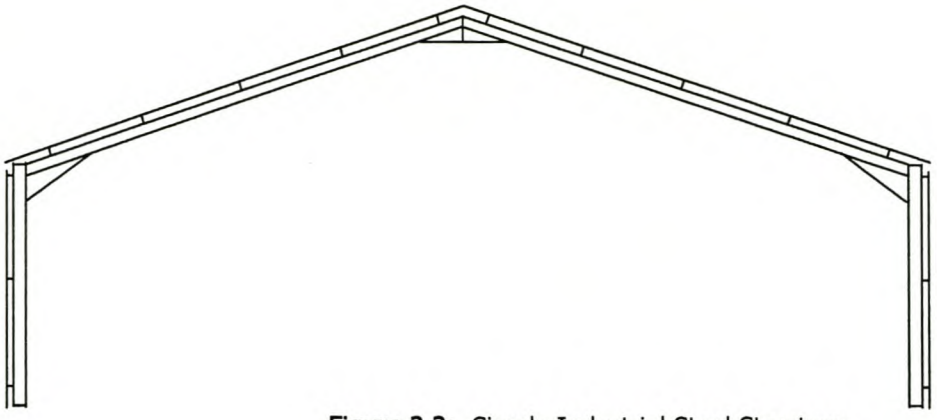


Figure 2.2: Simple Industrial Steel Structure

The first step in identifying the necessary subset of entities is to define the main structural components of the structure shown in Figure 2.2. The main structural components are subdivided into parts and connections. The beams, columns and sheets between the beams and columns are all structural parts. The joints between the beams and columns are the structural connections.

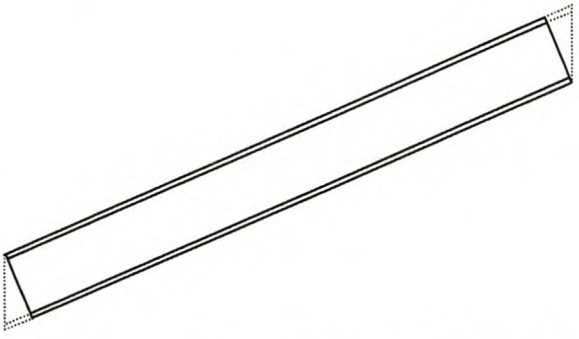


Figure 2.3: Beam of Simple Structure

Consider a beam from a simple structure as shown in Figure 2.3. The beam is identified from the STEP entities as a structural part (*SPart*). A structural part can be subdivided into three components. These are:

- Sheet parts
- Prismatic parts
- Pseudo prismatic parts

A beam is considered a prismatic part since it has a uniform geometry across its entire length. The difference between prismatic and pseudo prismatic parts are shown in Figure 2.4. Zones 1 and 3 are pseudo prismatic parts; their geometry changes along their length and does so in a uniform fashion, while zone 2 is a prismatic part.

The definition of a pseudo prismatic part is further shown in more graphical detail in Figure 2.5. This figure shows clearly how the geometry of a pseudo prismatic part changes in a uniform manner.

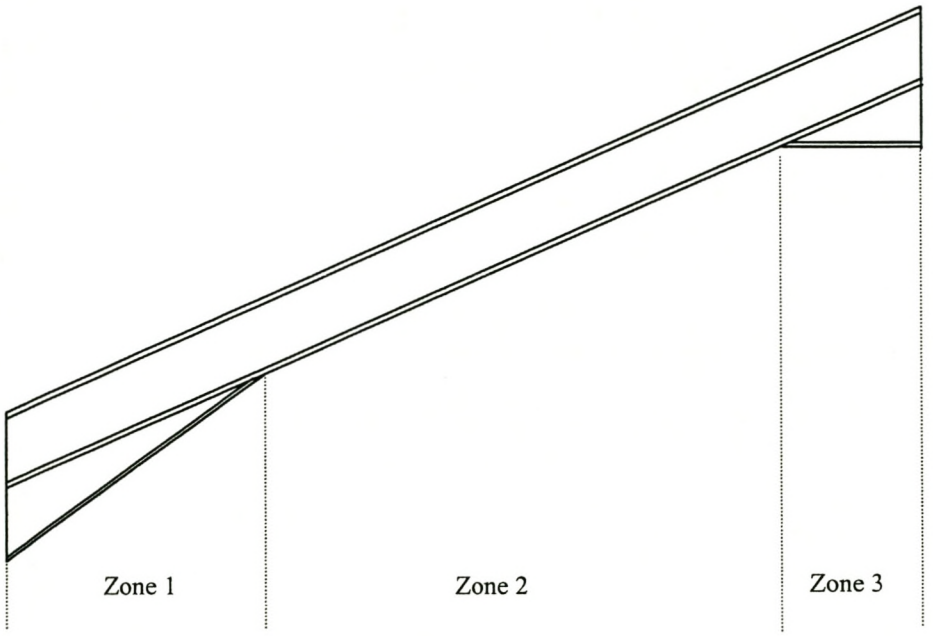


Figure 2.4: Details of a Structural part

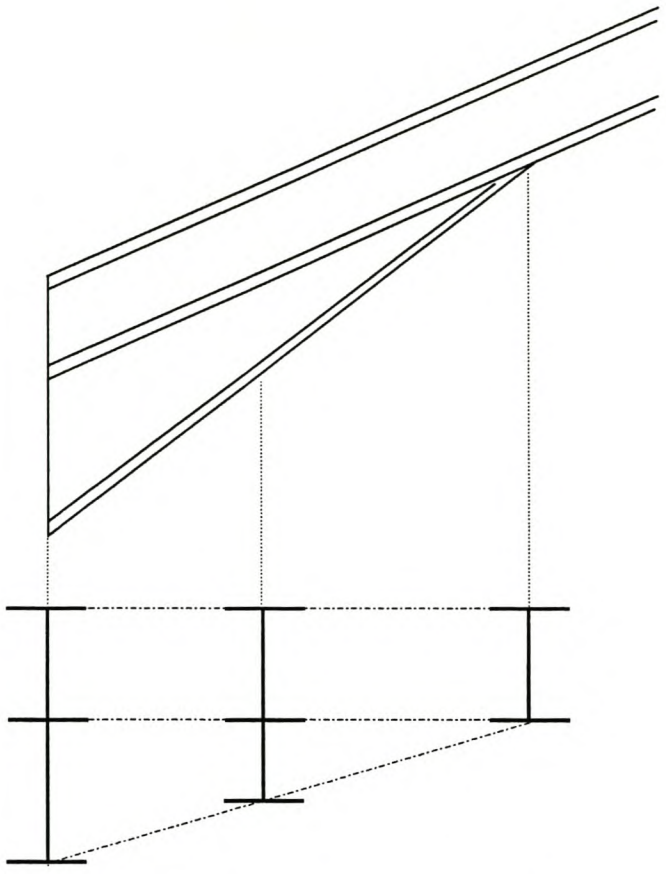


Figure 2.5: Demonstration of a Pseudo prismatic part

The geometry of the different structural parts is captured in two STEP entities. These entities are:

- SheetPartGeom
- PrismaticPartGeometry

SheetPartGeom describes the geometry of sheet parts only, while the entity *PrismaticPartGeom* describe both prismatic parts and pseudo prismatic parts. Having defined the geometry it becomes necessary to specify the type of section profile. This is done using the STEP entity *SectionProfile*. The section properties are specified using this entity.

The section properties (*SectionProperties*) are a collection of static characteristics related to the cross-sectional geometry of a steel member. From the entity, *SectionProfile*, it is possible to specify the type of section as either one of:

- Angle section

- Circle section
- I-type section
- Rectangular section
- T-type section

After the section type of the structural part is specified the next step in identifying the necessary STEP entities is to describe the ends of the beam. The ends under consideration are shown as a dashed line in Figure 2.3. These ends are described as structural features (*SFeature*) in the STEP entities. A structural feature is subdivided into four subsections. The entities describing these four sections are:

- CuttingPlane
- EdgeChamfer
- PrismaticEndFeature
- ProceduralFeat

Since a beam is a prismatic part, or even a pseudo prismatic part, the entity necessary here is a prismatic end feature. The end feature of the beam under consideration here is a skewed end.

Consider an example of a structural member with the dimensions shown in Figure 2.6. The information needed to describe this member is shown in Figure 2.7. The example is continued in Figure 2.8, where values of the structural feature are shown.

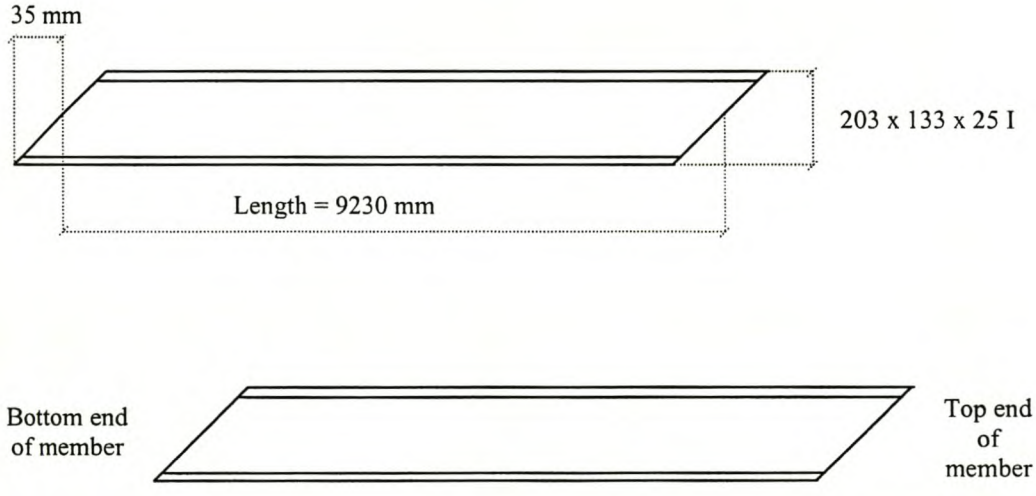


Figure 2.6: Dimension of an I-section member

The listed entities are also shown graphically in Figures 2.8 to 2.10. The relationships between the entities are also shown in these figures. From the relationships it can be seen how the STEP entities are linked to one another.

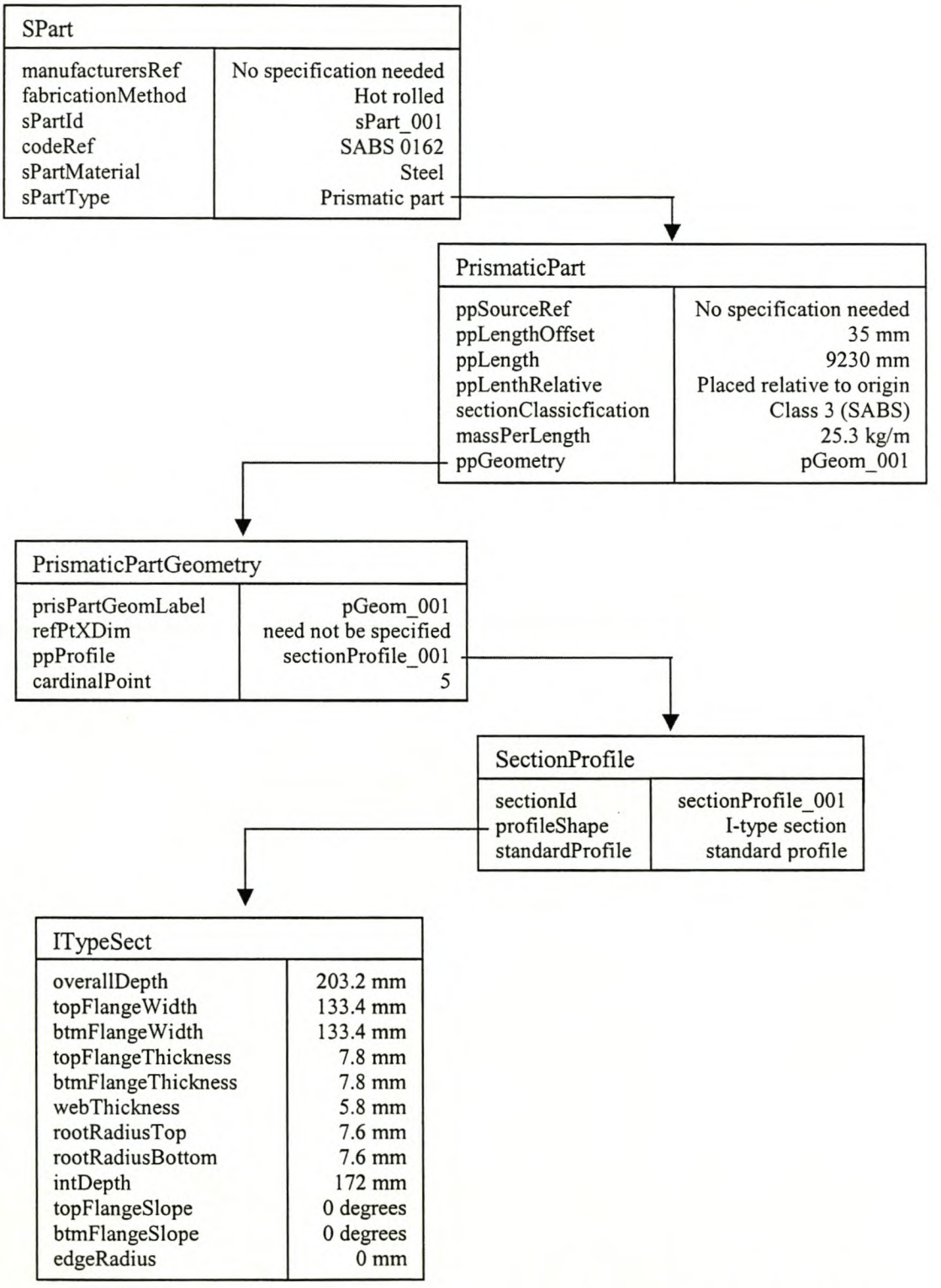


Figure 2.7: Entities with values needed to describe a Structural member

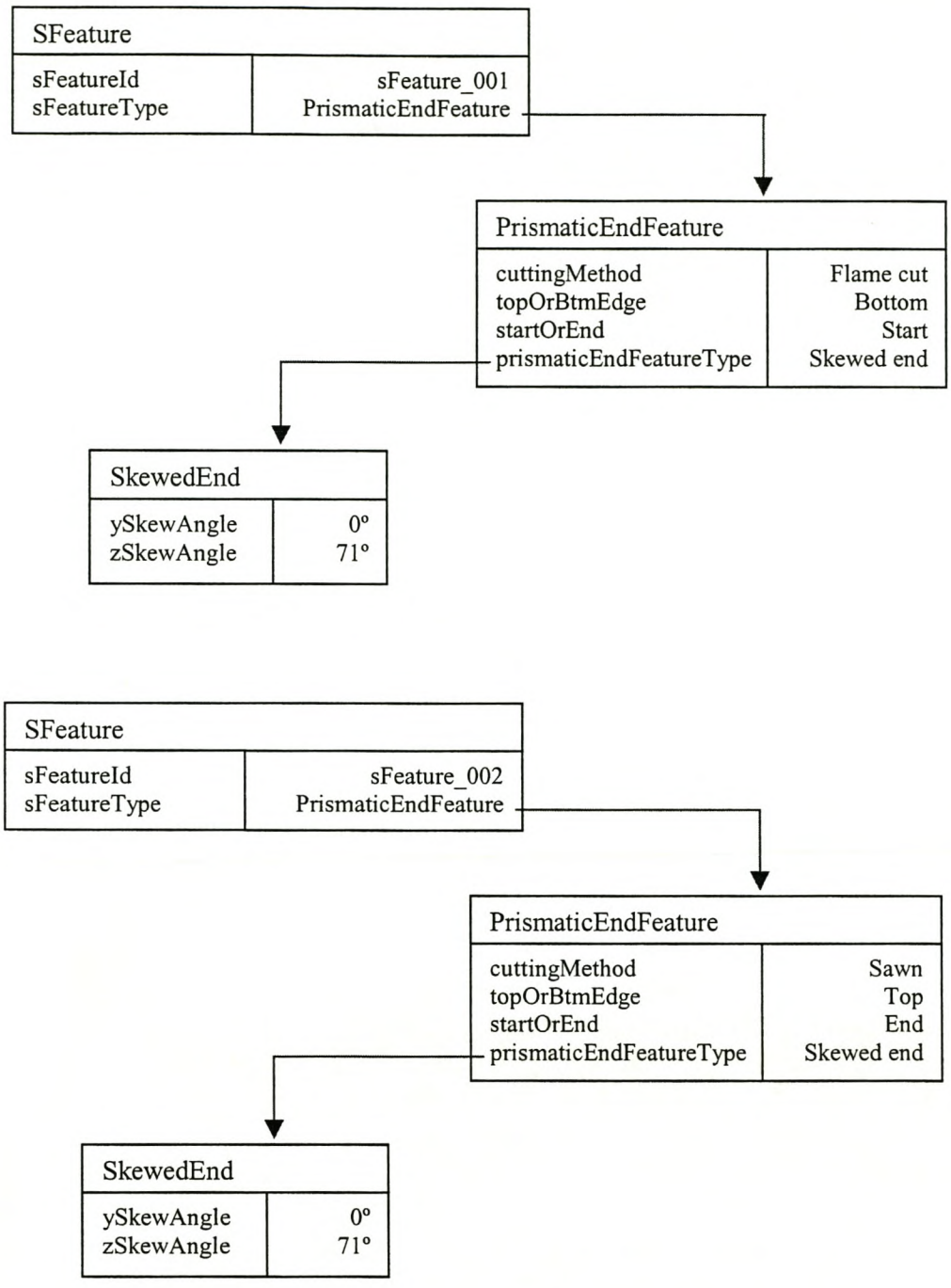


Figure 2.8: Entities with values needed to describe a Structural feature

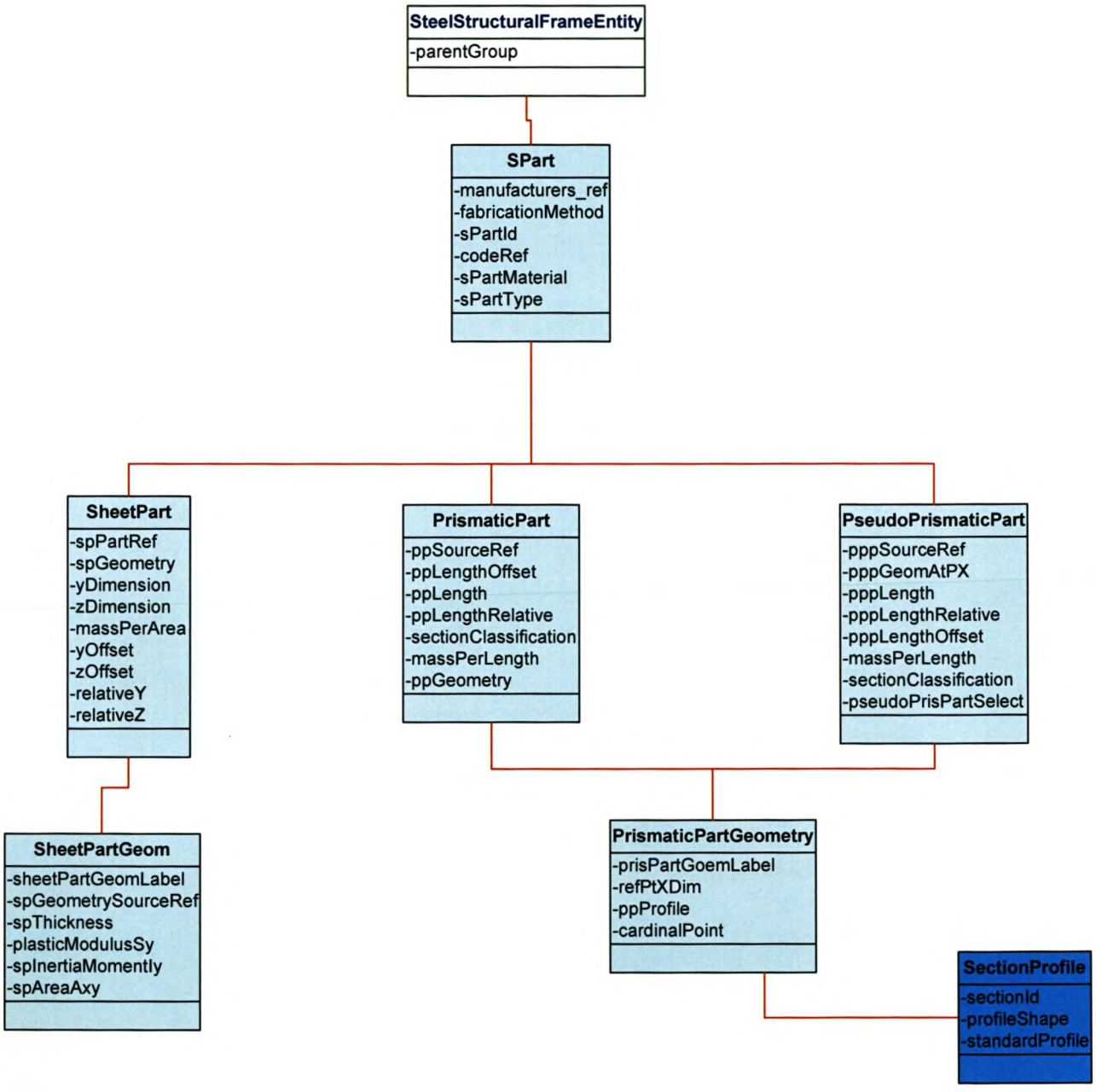


Figure 2.9: Layout of Structural Part

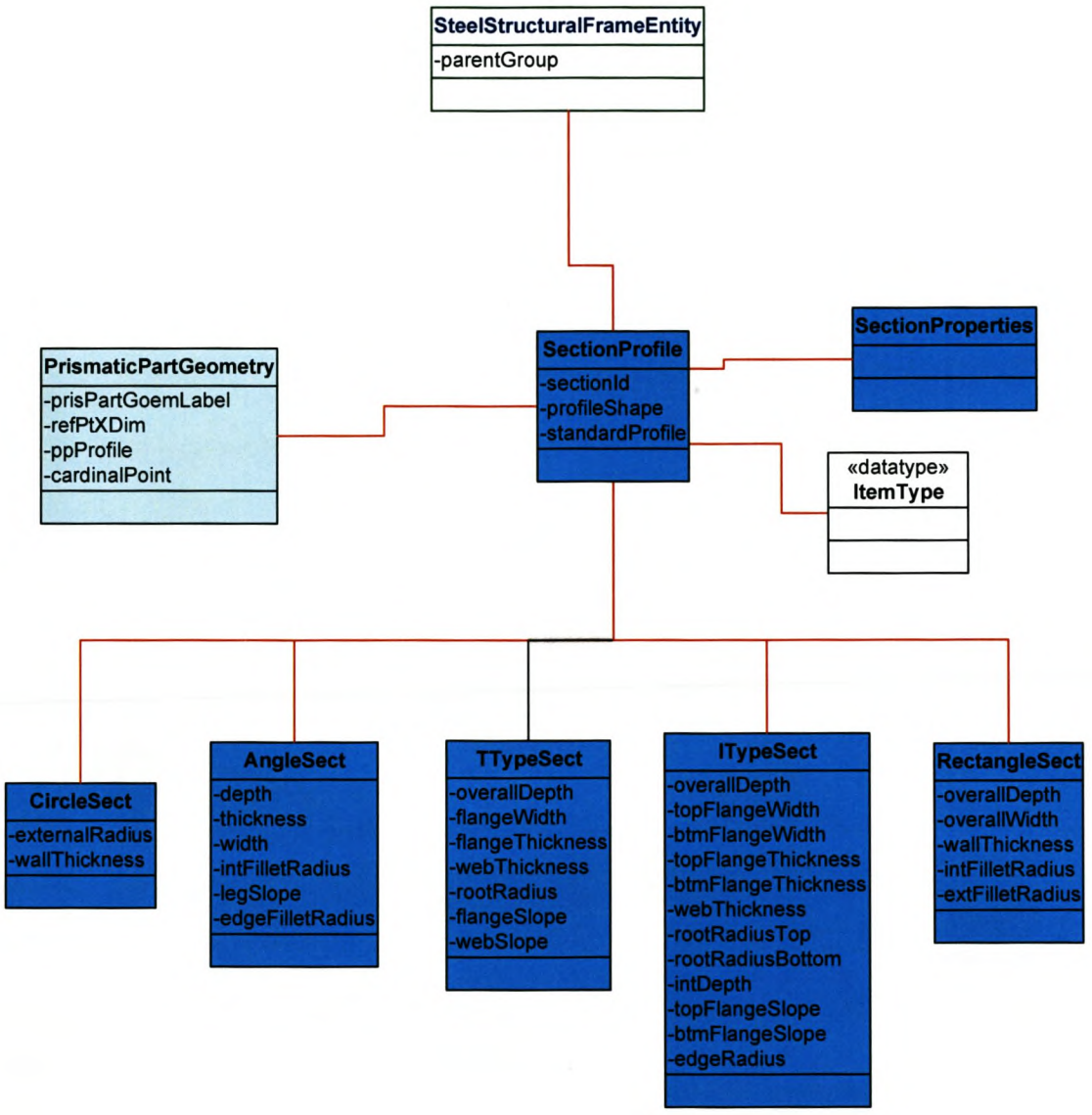


Figure 2.10: Layout of Section Profile

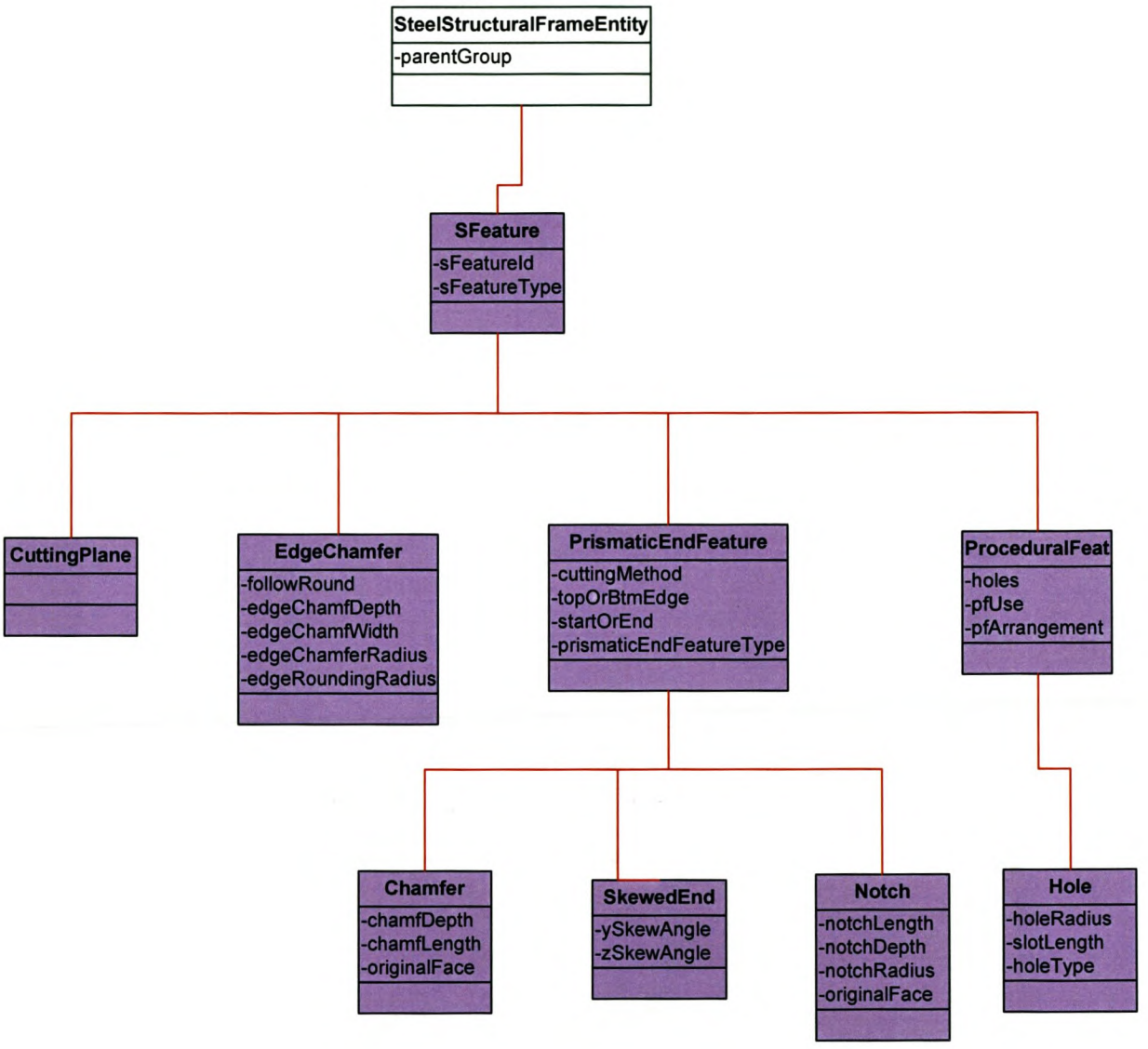


Figure 2.11: Layout of Structural Feature

2.2 Entities describing Structural Joints

Once a structural member has been defined it is necessary to define the joints connecting these structural members with one another. An example of the different types of joints (connections) that can be found in a simple industrial steel structure is shown in Figure 2.12.

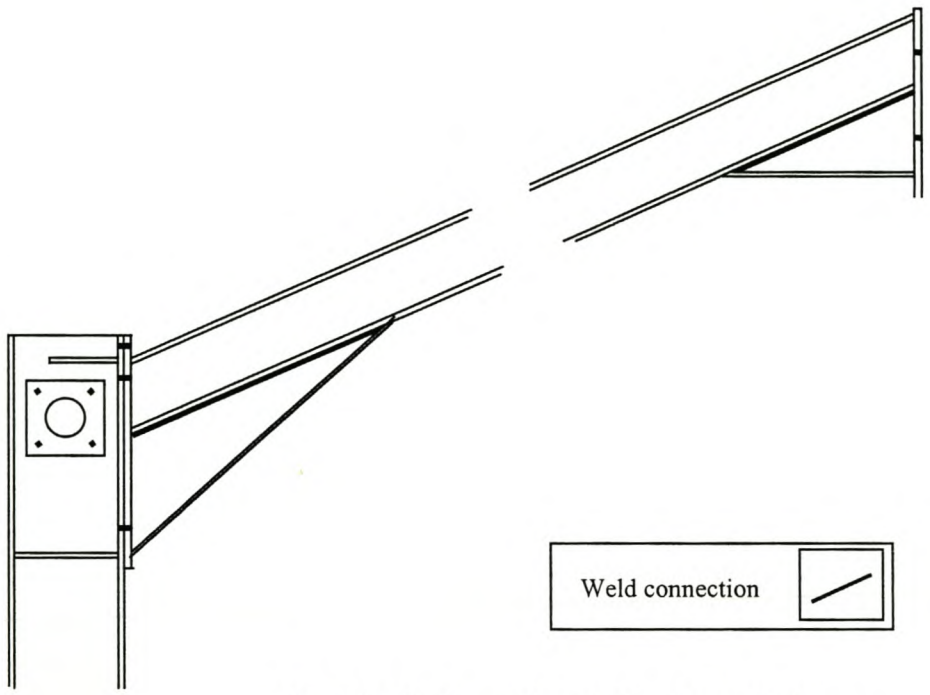


Figure 2.12: Connection details of a portal frame

The two type of joints identified are either bolted- or welded joints. These form subcomponents of the STEP entity structural joint system (*SJointSystem*). Bolted connections are made up of a bolt system (*BoltSystem*). This bolt system specifies the number of bolts in a bolted connection as well as their layout in the connection. Each bolt system is in turn made up of several mechanisms (*BoltMechanism*). Each bolt mechanism is made up of one or more bolts, nuts and washers. A typical bolt mechanism is shown in Figure 2.13.

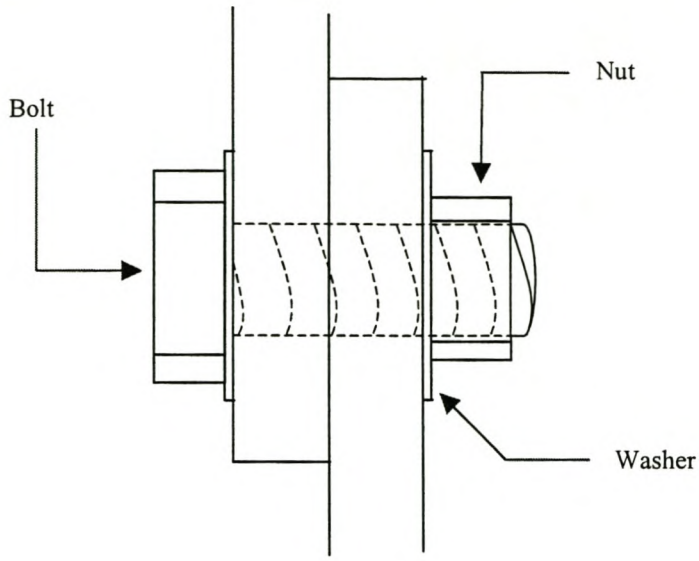


Figure 2.13: Bolt mechanism

A welded joint is treated similarly. It also consists of a system (*WeldSystem*), which specifies the number of welds and the path of the weld. Each weld system is made up of weld mechanisms (*WeldMechanism*). The welded joint in Figure 2.12 is enlarged and shown as a side view in Figure 2.14.

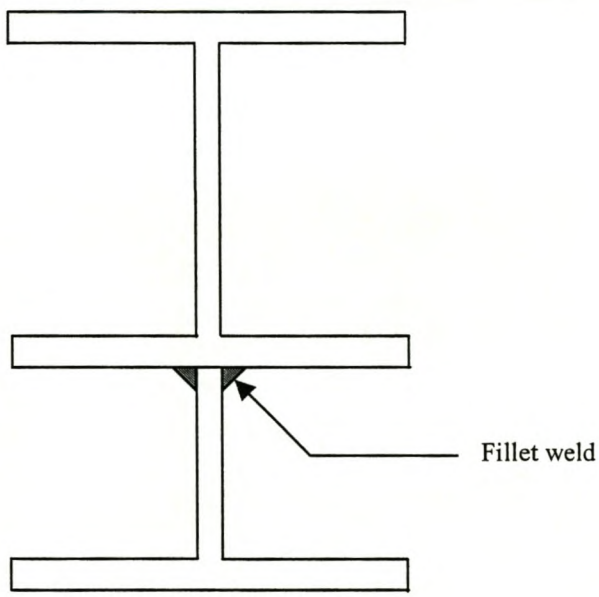


Figure 2.14: Weld mechanism forming a Pseudo-prismatic section

A representation of the entities, with example values, needed to define the joints of a steel structure is shown in Figure 2.15 and 2.16. Figure 2.15 is a demonstration of entities needed to define a bolted joint. Figure 2.16 defines a welded joint.

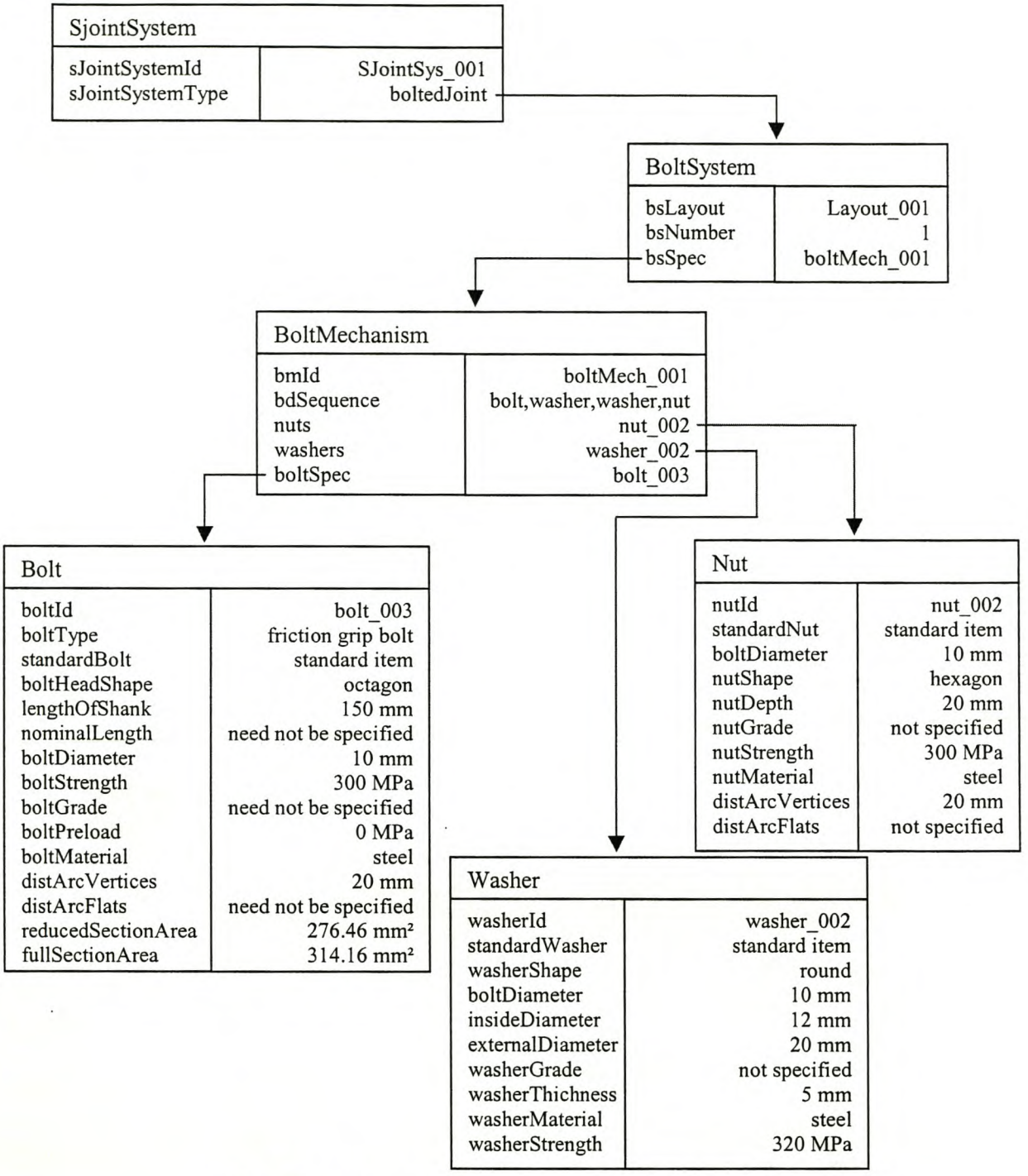


Figure 2.15: Entities with values needed to describe a Bolted joint

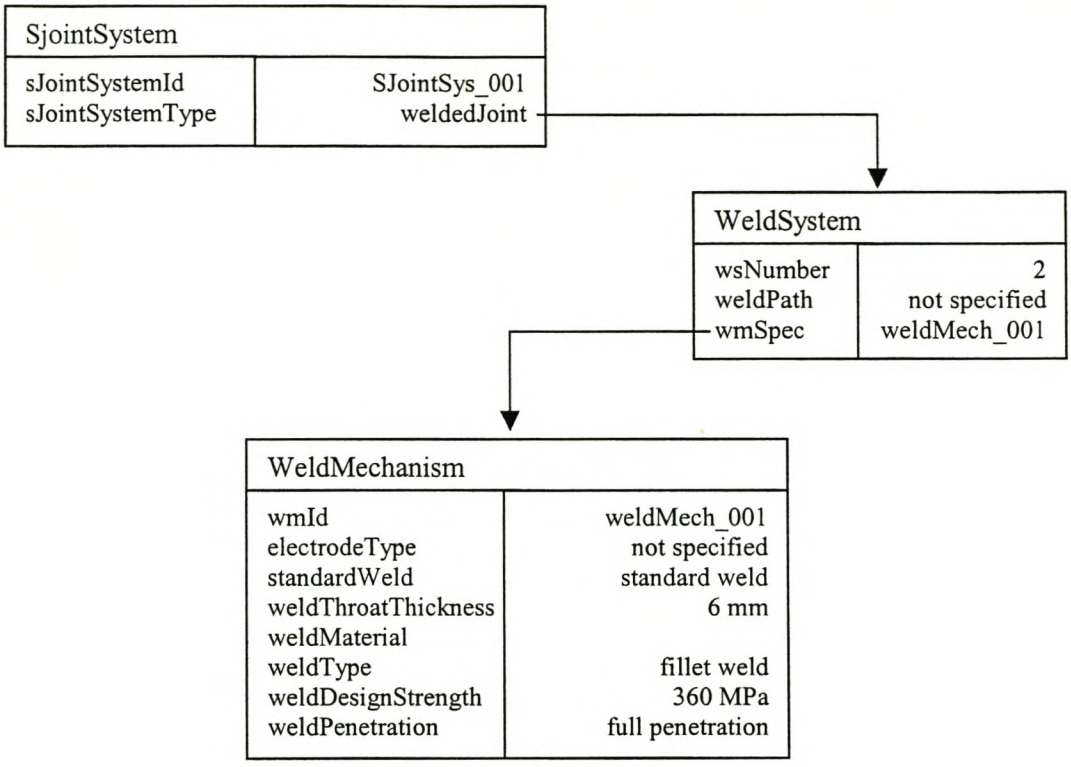


Figure 2.16: Entities with values needed to describe a Welded joint

The listed entities are also shown graphically in Figure 2.17. The relationships between the entities are also shown in this figure.

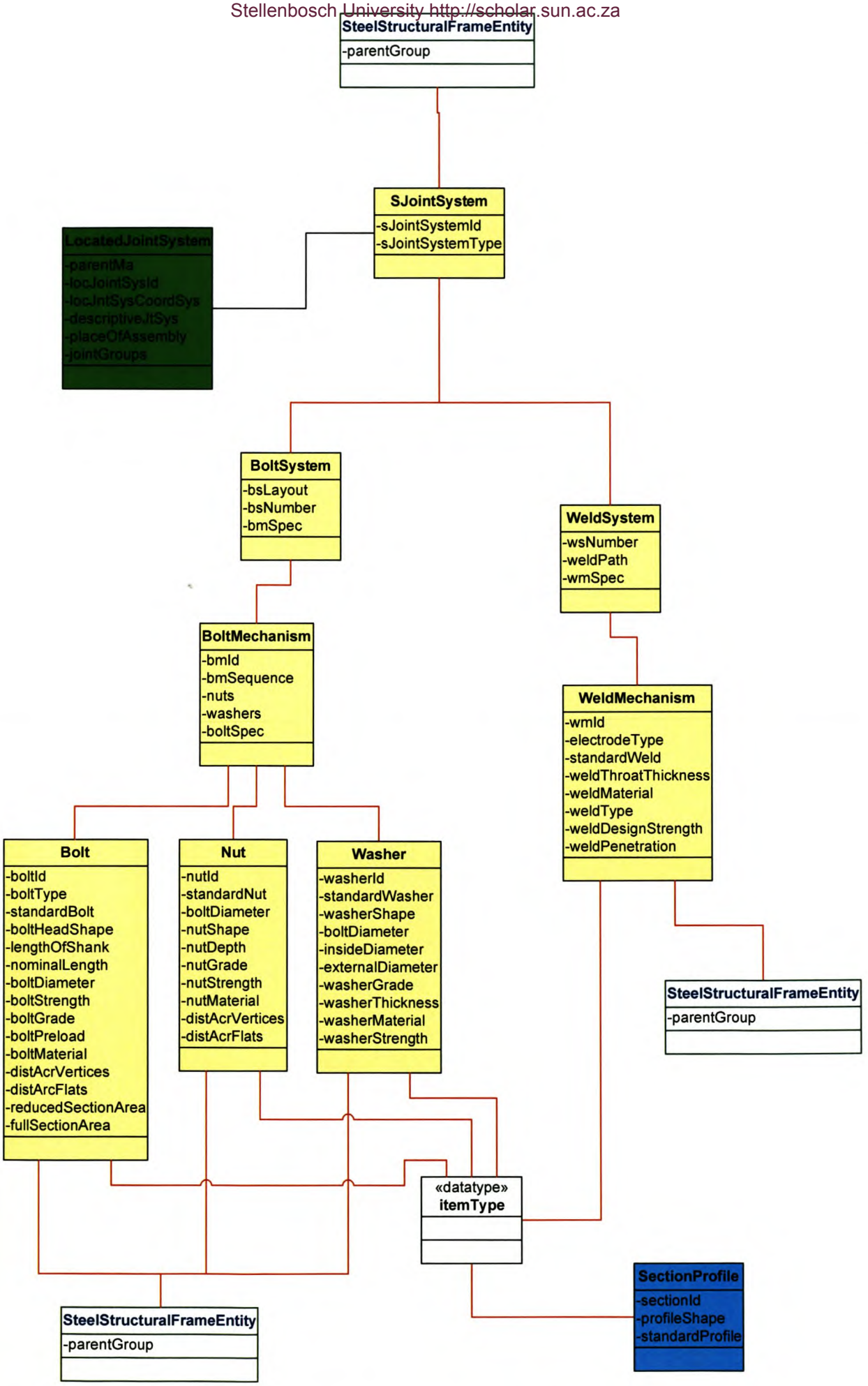


Figure 2.17: Layout of Structural Joint System

2.3 Entities linking information between Structural members and Structural joints

Having defined the STEP entities necessary to define both structural members and structural joints, it becomes important that there exists a means of linking information between them. It is important to do this otherwise a member is defined without a connection, or a connection is defined without the structural members which it connects. A typical connection of structural members is shown in Figure 2.18.

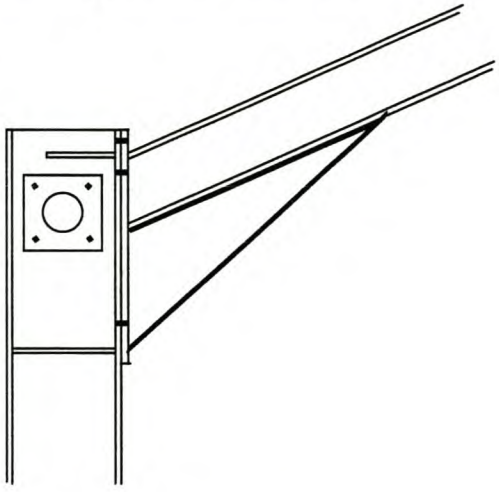


Figure 2.18: Structural members connected to each other with a joint

The linking entity *LocatedPart* is associated with a structural member. A located part is a structural part that has been located and oriented relative to a manufacturing assembly. A *PartJoint* forms the link between the linking entities *LocatedPart* and *LocatedJointSystem*. A *PartJoint* is a logical connection between a pair of located parts.

A *LocatedJointSystem* is a structural joint system that has been located and oriented to a manufacturing assembly (Figure 1.2) and thus to a structure (a discrete structural unit). The located parts involved in a *PartJoint* are physically connected by the application of one or more located joint systems.
A structural feature is described by the linking entity *LocatedFeature*. A located feature forms the link between a structural feature and a *JointDepFeature*. A *JointDepFeature* is a type of *LocatedFeature* that provides an association between a located feature and a located joint system. This entity is only necessary in situations where a feature is created as part of a joint.

An example of the STEP entities linking information between the already defined entities *SPart*, *SFeature* and *SJointSystem* is demonstrated in Figure 2.19.

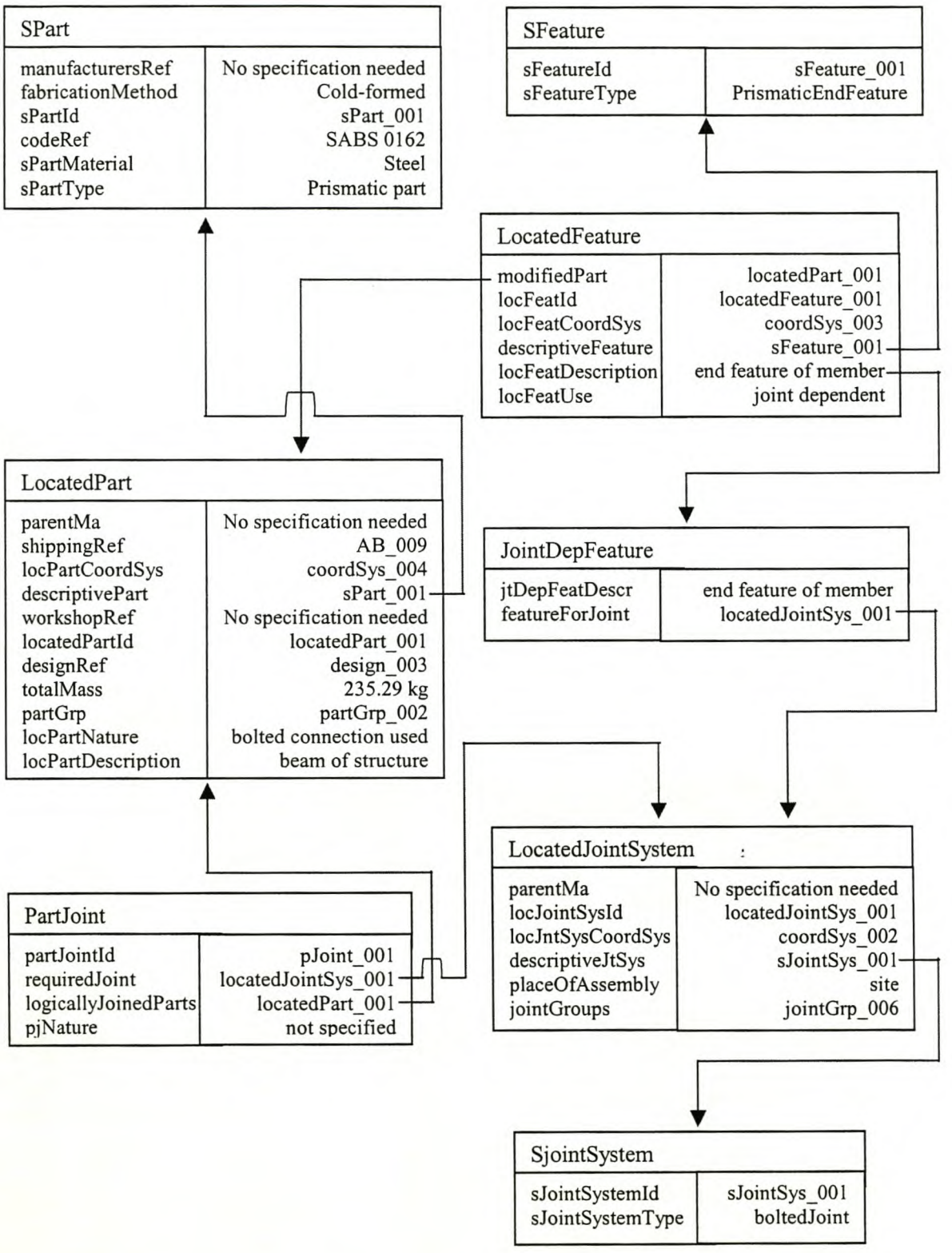


Figure 2.19: Entities with values needed to describe the linking of information

2.4 Entities describing Analysis information

Once the structural members and joints have been identified and described it becomes possible to analyse it for design purposes. A simple design representation of a structure is shown in Figure 2.20.

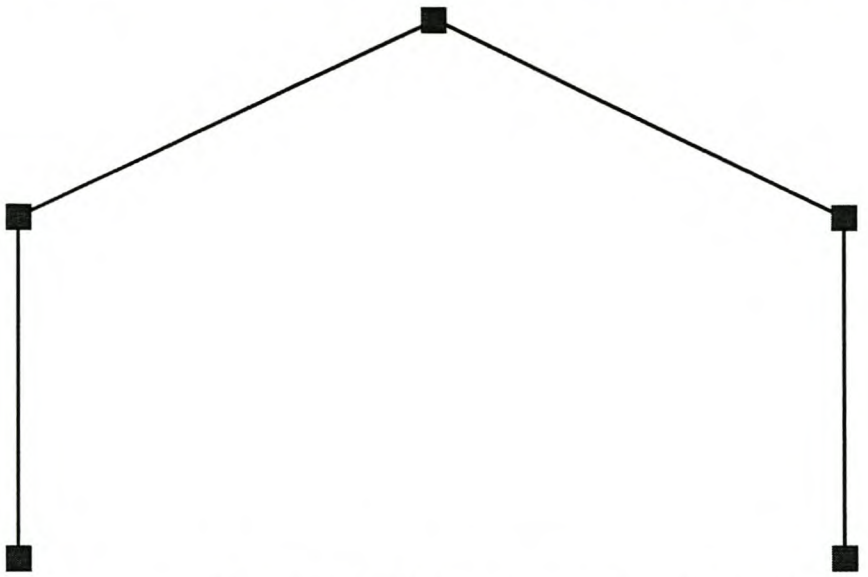


Figure 2.20: Design representation of a Steel structure

The entity needed to describe such a layout is called a *GeometricRepresentation*. A *GeometricRepresentation* is the geometrical characteristics of parts (*SPart*) and elements (*Element*). Each *GeometricRepresentation* is either a *PrismaticPartGeometry* or a *SheetPartGeom*. The description of the geometrical representation of a structural part as either a prismatic part or a sheet part was shown in Figure 2.9.

The elements in the wire frame shown in Figure 2.20 are described by the STEP entity *Element*. An *Element* is described as a one, two or three-dimensional entity that is used to model a part of a structure during structural analysis. Once an element is defined the next step is to describe the link between elements. The entity responsible for this description is the STEP entity *Node*. A *Node* is a theoretical point in space that is used in structural analysis to define a place where two or more elements meet and where a structure may be restrained. The Layout of a geometrical representation is shown graphically in Figure 2.21.

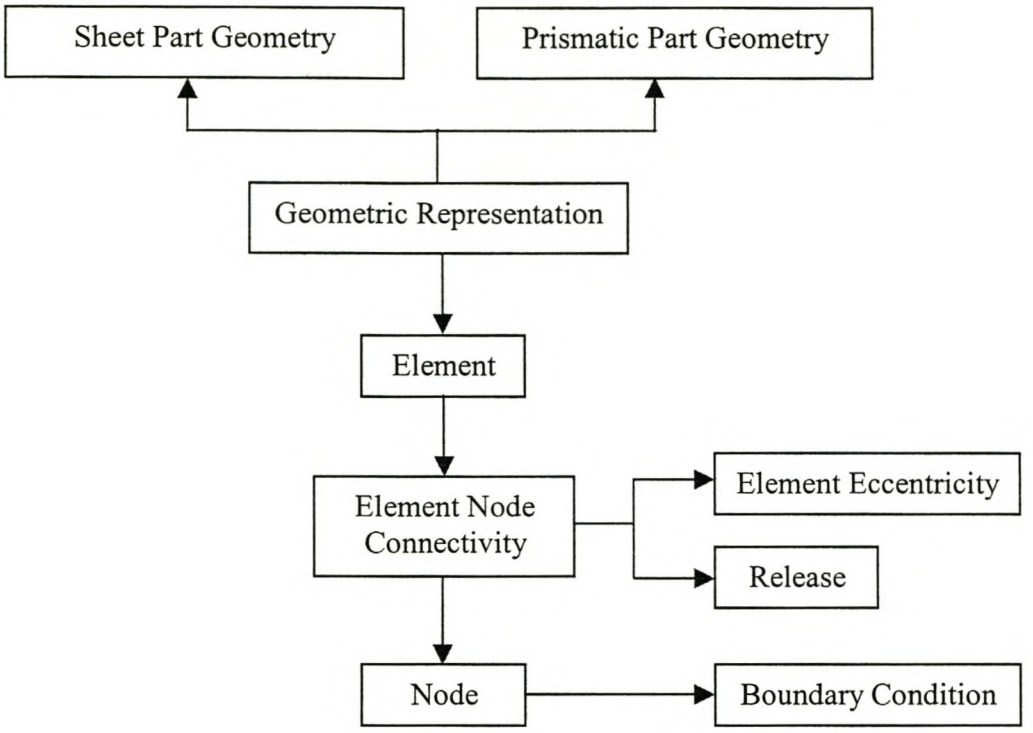


Figure 2.21: Layout of a Geometric representation

Once the main components of the wire frame have been defined with the entities nodes and elements it becomes necessary to describe them in more detail. The entities describing the nodes and the elements in more detail are:

- Element:
  - ElementEccentricity
  - Release
- Node:
  - Point
  - BoundaryCondition

A description of these entities are described in Table 2.1.

Model Description (Described by Entities)			Wire Frame Model
Structural Part	Element	An element is used to model a part of a structure during structural analysis	Describes the wire frame as a 2D or 3D entity used to model the structure. The element ends are at the centres of the connecting nodes.
		Element Eccentricity	Is used when modelling elements whose end-centres do not coincide with their connecting nodes
		Release	Defines the degrees of freedom of an element at the point where it connects with a node
	Node	A node defines a theoretical point in space that is used in structural analysis to define a place where two or more elements meet and where a structure may be restrained	Defines the link (connection) between elements. It is defined as a point in space. Element centrelines co-inside with the node centres.
		Point	A point defines a node as a specific location in a 2D or 3D space, by means of the global coordinates system of the model.
		Boundary Conditions	Describes information relating to the 6 degrees of freedom, 3 translations and 3 rotations of a supporting node.

Table 2.1: Geometrical representation of a Structural Part

The STEP entity *ElementEccentricity* is used to model elements whose end-centres do not coincide with their connecting nodes, while the STEP entity *Release* defines the degrees of freedom of an element at the point where it connects with a node. These entities describing the geometry of a structural part are shown in Figure 2.22.

The STEP entity *Point* is a location in one, two, or three-dimensional space defined by Cartesian coordinates. The coordinate system is further defined in Figure 2.23.

The nodes of the wire frame also describe the boundary conditions. The entity *BoundaryCondition* holds information relating to the 6 degrees of freedom, 3 translations and 3 rotations, of a supporting node in an analysis model.

As mentioned above, Cartesian coordinates define a *Point*. This is done with the STEP entity *CoordSystem*. This entity defines a 3-axis Cartesian coordinate system by reference to a parent or (ultimately) a global coordinate system. With the coordinate system laid out, it is possible to transform the coordinate system from a local coordinate system to a global coordinate system. This is done with the STEP entity *Transformation* which locates one coordinate system with respect to another.

There are three other STEP entities that need mentioning when the coordinate system of a structural representation has been defined. These entities are:

- Gridline
- GridlineSet
- Layout

A *Gridline* defines a line (of infinite length) that passes through two specified points in a horizontal plane. This line is used as a reference when locating structural members and connections. A *GridlineSet* defines a set of uniform or non-uniform gridlines.

A *Layout* is a set of reference points lying in a plane. The points defined in a coordinate system from a specific layout lie in the yz plane of that system.

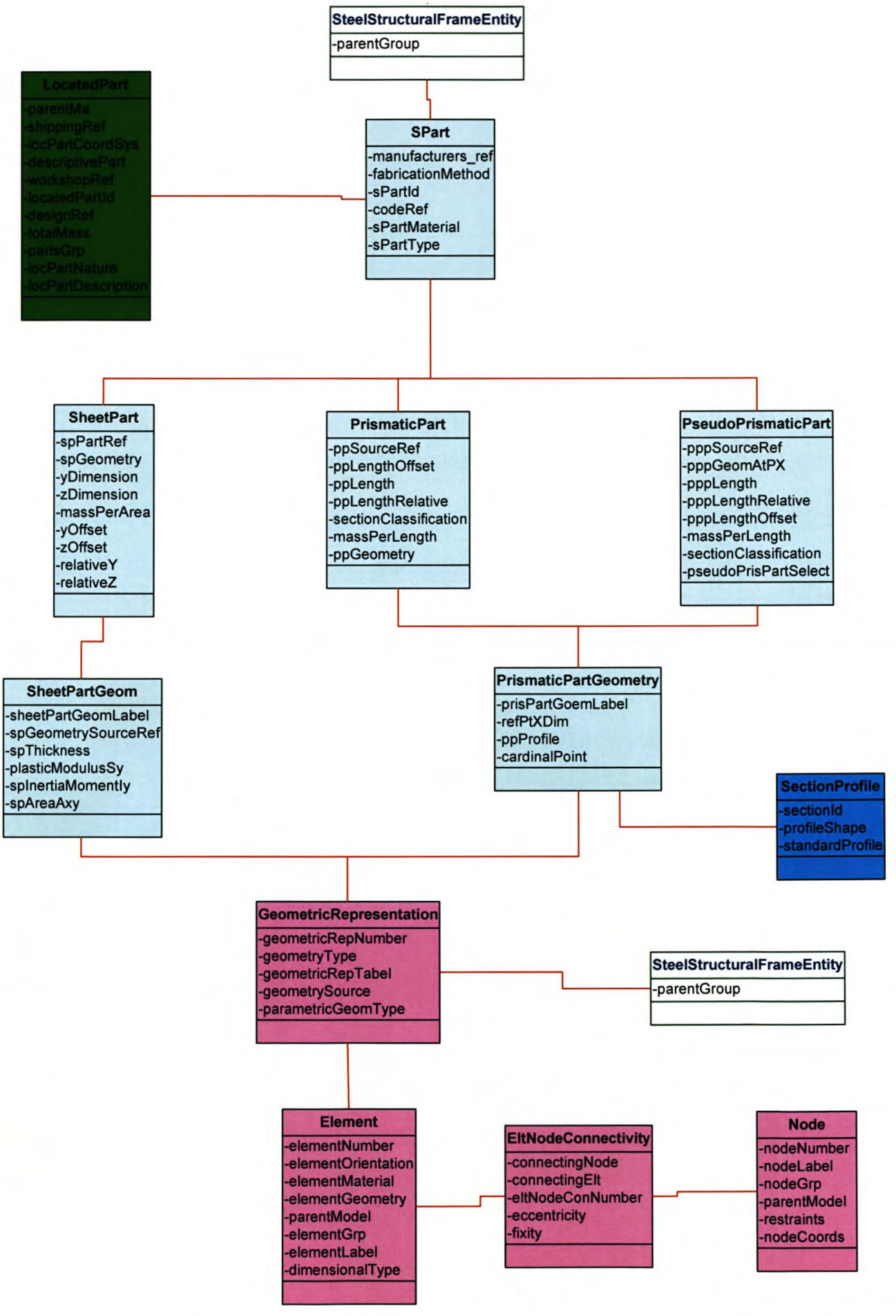


Figure 2.22: Layout of completed Structural Part

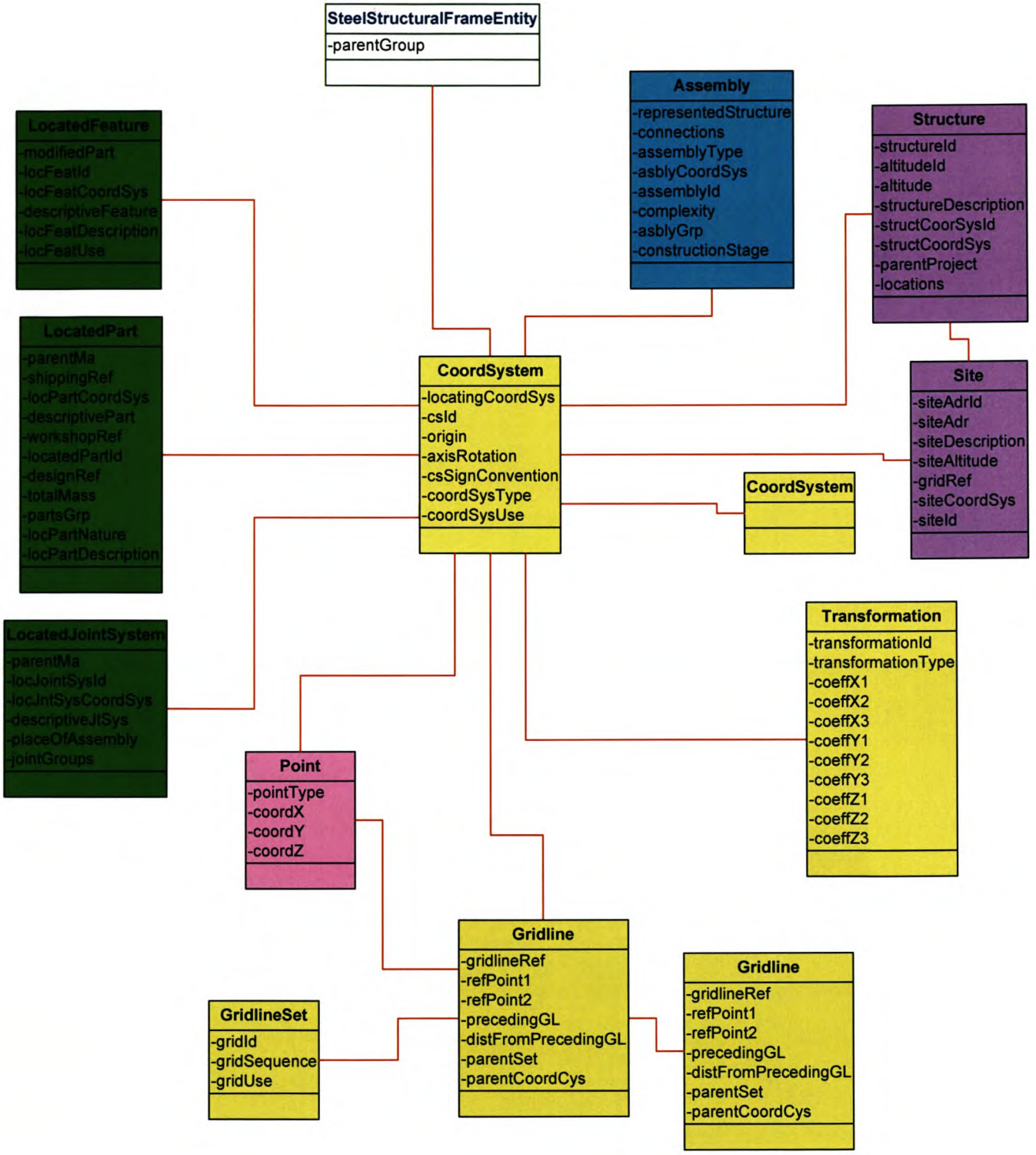


Figure 2.23: Layout of Coordinate System

2.5 Entities for Assembling members and joints

A layout of the STEP protocol for the design assembly of a simple industrial steel structure is shown in Figure 2.26. A design assembly is the description of the structural parts and structural joint systems of a data model from a design point of view. The collection entity defining all structural parts in a design assembly is a design part. The entity defining all structural joint systems is a connector.

A design part (*DesignPart*) is a representation of a physically located part for member design purposes. Its representation may or may not correspond directly to an analytical element (*Element*). A connector holds information relating to the design of welded or bolted connections. The layout of the design hierarchy is shown in Figure 2.24.

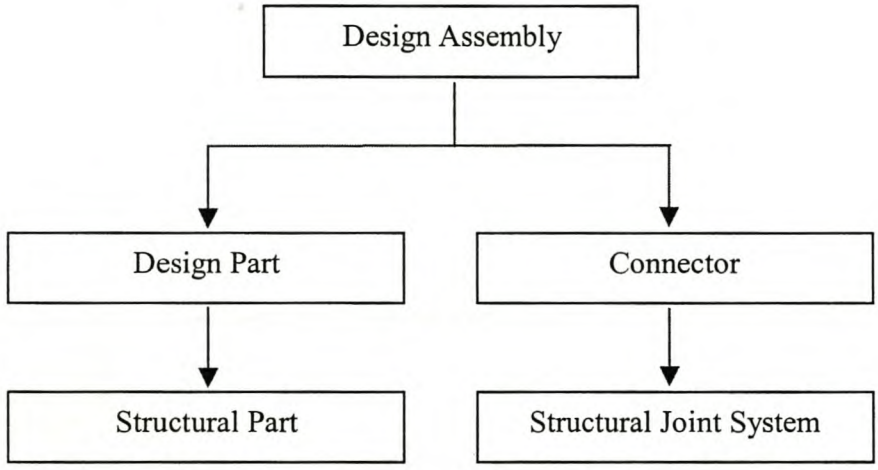


Figure 2.24: Simple Industrial Steel Structure

The hierarchy in Figure 2.24 shows the entities arranged in a graded series. The most important entity needed to assemble members and joint is the design assembly entity. From this entity both members and joints are described and linked.

The different colours in Figure 2.25 are used to show how the different STEP entities defined in the previous sections fit together. The parts are set out in Figure 2.26:

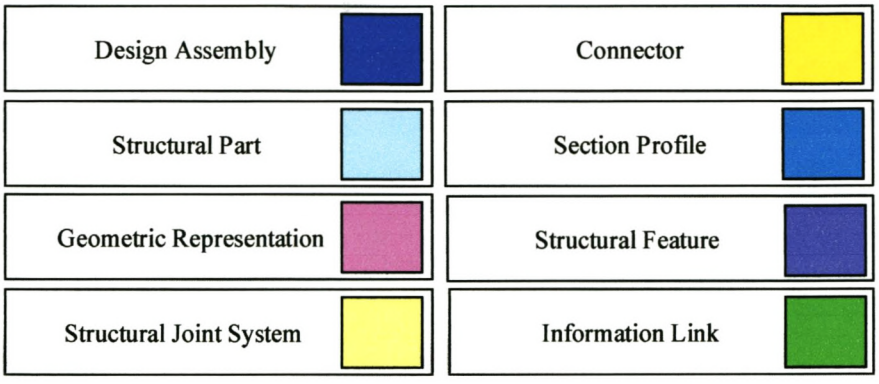


Figure 2.25: Key to Design Assembly diagram

Figure 2.26: Layout of Design Assembly (Identified STEP Entities)

2.6 Format of entity description in Appendix A

All the identified subsets of entities necessary to define a simple industrial steel structure are described in Appendix A. An example of how to interpret such an entity is presented below.

The entity under discussion is a node (Appendix A, number 13). The entity is first defined:

- A node is a theoretical point in space that is used in structural analysis to define a place where two or more Elements meet and where a Structure may be restrained.

The definition is followed by the data (attributes) associated with the entity. Only a few of the attributes associated with the entity node is shown here. These are:

- nodeNumber
  - a unique numeric identifier for a Node
- nodeLabel
  - short text description, used in conjunction with nodeNumber to uniquely identify a Node
  - need not be specified for a particular Node
- parentModel
  - an analysis model to which a Node belongs
  - need not be specified for a particular Node
  - application assertion
  - Node is a component of one AnalysisModel
  - each AnalysisModel includes one or many Node objects

Some of the attributes are not absolutely necessary to define an entity and need not be specified. An example of such an attribute is *nodeLabel*, it is therefore left to the user to decide whether or not to enter values for such attributes. Such attributes are therefore included in the description of an entity to accommodate the level of information that can be stored. This means that if more information is required about a specific entity, these attributes are used by the user to include the additional information.

Some of the attributes are used as links to other entities. Investigating the attribute *parentModel* it is clear that by definition a node is a component of an analysis model. The value of *parentModel* links the node to the specified analysis model.

2.7 List of all selected entities

The selected entities were identified from AP230 by the author. The entities were identified since they are the primary entities needed to describe a simple industrial steel structure. The paragraphs below place these entities in the category where they are used, i.e. to describe:

(i) Structural Members

(ii) Structural Joint Systems

(iii) Information linking Structural Members and Structural Joint Systems

(iv) Analysis.

2.7.1 Entities describing a Structural member:

All the entities that are needed to identify and represent a structural member such as a beam, column or plate are listed below. These entities are described in great detail in Appendix A, parts 1 to 35.

The hierarchy of the identified STEP entities needed to describe a structural member are:

- DesignAssembly (1)
- DesignPart (2)

Entities describing a *SPart* (Appendix A, part 3) are:

- SheetPart (4)
- PrismaticPart (6)
- PseudoPrismaticPart (8)
- SheetPartGeom (5)
- PrismaticPartGeometry (7)

Entities needed to describe a *SectionProfile* (Appendix A, part 20) are:

- SectionProperities (21)
- AngleSect (22)
- CircleSect (23)
- ITypeSect (24)
- TtypeSect (26)
- RectangleSect (25)

The entities needed to describe a *SFeature*(Appendix A, part 27) are:

- CuttingPlane (28)
- EdgeChamfer (29)
- PrismaticEndFeature (30)
- ProceduralFeat (34)
- Chamfer (31)
- SkewedEnd (32)
- Notch (33)
- Hole (35)

2.7.2 Entities describing a Structural Joint Systems:

All the entities that are needed to identify and describe a structural joint consisting of either a bolted or welded joint are listed in this section. The identified entities needed to define a structural joint are defined in detail in Appendix A, parts 43 to 51. The hierarchy of these identified STEP entities needed to describe a structural joint is:

- Connector (43)

Entities describing a SJointSystem (Appendix A, part 44) are:

- BoltSystem (45)
- WeldSystem (50)
- BoltMechanism (46)
- WeldMechanism (51)
- Bolt (47)
- Nut (48)
- Washer (49)

2.7.3 Entities used to link information between Structural Members and Structural Joint Systems:

These entities discussed above are described in detail in Appendix A, parts 36 to 40. These entities necessary to describe the link between the members and joints are:

- LocatedPart (36)
- PartJoint (37)
- LocatedJointSystem (38)
- LocatedFeature (39)
- JointDepFeature (40)

2.7.4 Entities describing Analysis information:

All the entities that are needed to identify and represent the analysis of a structural representation are listed below. These entities are described in great detail in Appendix A, parts 9 to 19 and part 41.

The main STEP entities needed to define the analysis of a structure are shown below together with the additional entities needed to describe them. The entities are:

- GeometricRepresentation (9)
- Element (10)
  - ElementEccentricity (11)
  - Release (12)
- Node (13)
  - Point (14)
  - BoundaryCondition (19)

Entities describing the coordinate system and the layout of points are:

- CoordSystem (15)
- Transformation (16)
- Gridline (17)
- GridlineSet (18)
- Layout (41)

CHAPTER 3

DATABASE MODEL OF THE IDENTIFIED STEP ENTITIES

3.1 Database technology overview

There are two different types of databases that can be implemented to store the identified STEP entities. The two database management systems considered are:

- Object database management systems (ODBMS), and
- Relational database management systems (RDBMS).

There are also a certain number of properties needed to define an adequate system. These properties are the following [10]:

- Atomicity
  - Results of a transaction's execution are either all committed or all rolled back. All changes take effect, or none do.
- Consistency
  - The database is transformed from one valid state to another valid state. This defines a transaction as legal only if it obeys user-defined integrity constraints. Illegal transactions aren't allowed and, if an integrity constraint can't be satisfied then the transaction is rolled back.
- Isolation
  - The results of a transaction are invisible to other transactions until the transaction is complete.
- Durability
  - Once committed (completed), the results of a transaction are permanent and survive future system and media failures.

To choose the appropriate database system, bearing in mind the properties mentioned above, one has to look at the different systems. Firstly ODBMSs are considered followed by RDBMSs.

3.1.1 Object-Oriented Database:

Object systems contributed to software reliability and compactness by allowing programmers to factor their code into classes that are used as widely as possible.

An ODBMS persistently store the types of objects and pointer structures that one would create in a Java program. Chasing pointers and certain kinds of transactions can be 10 to 100 times faster that in a relational database [11].

Object databases have a big disadvantage since the programmer has to know a lot about the details of data storage. If the identities of the required objects are known, then the query is fast and simple.

There is no official standard for object databases. The emphasis of ODBMS is on direct (one-to-one) correspondence between the following [17]:

- The objects and object relationships within an application written in object-oriented languages
- The storage of those in the database

Object databases employ a data model that has object-oriented aspects like:

- classes, with attributes and methods and integrity constraints
- it provides object identifiers (OIDs) for any persistent instance of a class
- supports encapsulation (data and methods)
- multiple inheritance
- and supports abstract data types.

Object databases combine the elements of object orientation and object-oriented languages with database capabilities. They provide more than persistent storage of programming language objects. Object databases extend the functionality of object programming languages (e.g., C++, Java) to provide full-featured database programming capability. The result is a high level of congruence between the data model for the structures, and better maintainability and reusability of code.

Object database management systems (ODBMS) are defined and implemented explicitly to provide efficient storage of object-oriented applications.

3.1.2 **Relational Database:**

A relational database stores data in one or more tables, and these tables can be joined in a variety of ways to efficiently access information. A relational database is similar to a spreadsheet that several people can update simultaneously. A RDBMS manages tables with rows and columns. The rows correspond to a record (tuple); the columns correspond to the attributes (fields in the record). Each table in the database can be viewed as a spreadsheet.

A RDBMS is more restrictive than a spreadsheet in that all the data in one column must be of the same type. The type of data that can be stored are confined to a very limited number of data types, e.g.,

- integer
- decimal
- character string, or
- date.

Any attribute (field) of a record can store only a single value. Variable length fields are not supported.

A difference between a spreadsheet and an RDBMS is that the rows in an RDBMS are not ordered. If you define a *nodeCoords* column or some other unique identifier for rows in a table, it becomes possible for a row in another table to refer to that row by including the value of the unique ID.

The Primary Key is a column in a table that contains a unique value in every row. Every table in a database needs one of these, something that makes each row distinct. The primary key tells the database that this column's value can be used to uniquely identify a row.

Relationships are not explicit, but rather implied by values in specific fields: foreign keys in one table that match those of records in a second table. Many-to-many relationships typically require an intermediate table that contains just the relationships between tables.

Example:

In Figure 3.1 the attribute *nodeCoords* in table Node has the same value as the attribute *pointId* in table Point. These are linked enabling the coordinates of the node to be stored when the node itself is stored.

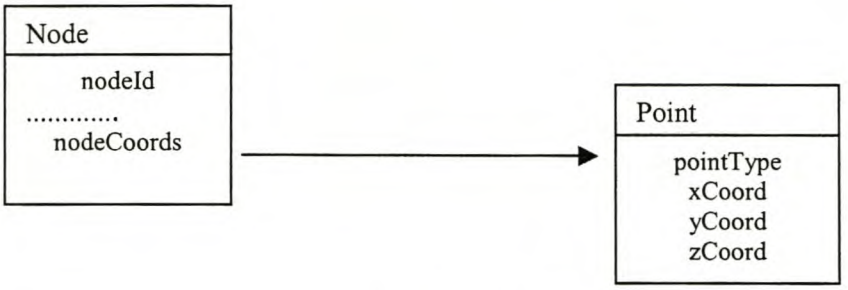


Figure 3.1: Referencing Tables

In Figure 3.1 the attributes *nodeId* and *pointType* are persistent identifiers (pid) used to uniquely identify a row. That means the system will reject an attempt to insert a row with the same pid as an existing row. This feature can have some unexpected performance implications. For example, every time anyone tries to insert a row into this table, the RDBMS will have to look at all the other rows in the table to make sure that there isn't already one with the same pid.

In Table 3.1 the database technology is summarised.

RELATIONAL DATABASE	OBJECT-ORIENTED DATABASE
Stores data in one or more tables, which can be joined in a variety of ways to efficiently access information.	Code is factored into classes, which can be widely used.
Data is stored with persistent identifiers.	Objects and pointers are persistently stored.
Data is stored within the database consisting of tables of rows and columns.	Combines object orientation and object-oriented languages with database capabilities.
Many-to-many relationships require an intermediate table that contains just the relationships	Extends the functionality of object programming languages.
Relationships are not explicit, but rather implied by values in specific fields.	ODBMS are defined explicitly to provide efficient storage.

Table 3.1: Database Technology

As was discussed in the previous section, ODBMS are defined and implemented explicitly to provide efficient storage of object-oriented applications. But for the implementation of the STEP entities the interest was not on object identities, the focus was more on the object attributes.

Even though object-oriented databases are very appropriate for managing complexity, relational databases tend to be faster and better at returning aggregations based on attributes.

The primary purpose of the database used in the collaborative design infrastructure is to support exchange of information. Therefore a relational database was selected for implementation of the STEP protocol. There are very few relations needed between the different tables. It is therefore possible to change the contents of a database table without having to change all the relations between the tables.

The database was developed in Microsoft Access, and is called Project Steel Structures (PSS).

3.2 Structure of a Standard entity table

The identified STEP entities discussed in Section 2.7 and shown in Figure 2.25 are all implemented in the relational database Project Steel Structures (PSS). A tables within PSS correspond to each entity. For each attribute of an entity, a column was created in the corresponding table. In addition to the stated attributes seven additional attributes were added at the beginning of each table. These attributes allow for versioning and access control.

Two attributes are common to all parameter classes, namely the persistent identifier (pid) and the version. Versioning of an entry is done by three attributes, the *versionNumber*, *versionTime* and *versionHistory*. In addition, for access control purposes, the database contains information about each parameter's owner, the task group the owner belongs to, and the access control code of the parameter.

Attributes	Description
pid	persistent identifier of an entry in the table
versionNumber	is the current version number
versionTime	is a version timestamp
versionHistory	history of a previous version numbers as a concatenated string
ownerId	current owner identification
taskgroupId	identifier of the taskgroup
accessCode	access control code of the entry

Table 3.2: Variables present in all database tables

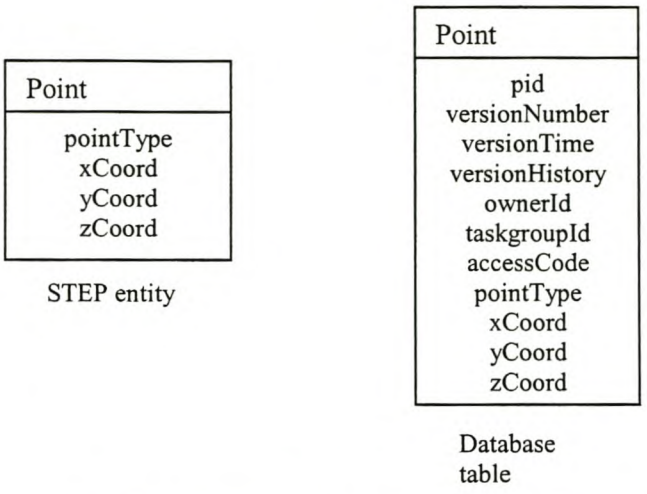


Figure 3.2: STEP entity expanded to a Database table

All the attributes that are represented in the identified STEP entities are retained. The seven common variables containing the pid, versioning and access control is entered first in the specified columns while the retained attributes of the specific entity follows. Therefore all the identified STEP entities, specified in Appendix A, are represented inside the database with all their attributes forming the columns of the tables.

SPECIAL CASE:

One STEP entity, namely *EltNodeConnectivity*, is not represented in the database since another table captures its data. The STEP entities *Element* and *EltNodeConnectivity* are combined to form a single entity, *Element*. Therefore a direct link between tables *Element* and *Node* is established. All the original attributes of entity *Element* are retained and six additional attributes are added to it. These six additional attributes are listed in Table 3.2 and the combination of the two STEP entities is also shown graphically in Figure 3.3.

Attributes	Description
pidNode1	persistent identifier of the starting node of the element
eccentricity1	eccentricity with respect to starting node of the element
fixity1	release associated with the starting node of the element
pidNode2	persistent identifier of the end node of the element
eccentricity2	eccentricity with respect to end node of the element
fixity2	release associated with the end node of the element

Table 3.3: Additional attributes in STEP entity Element

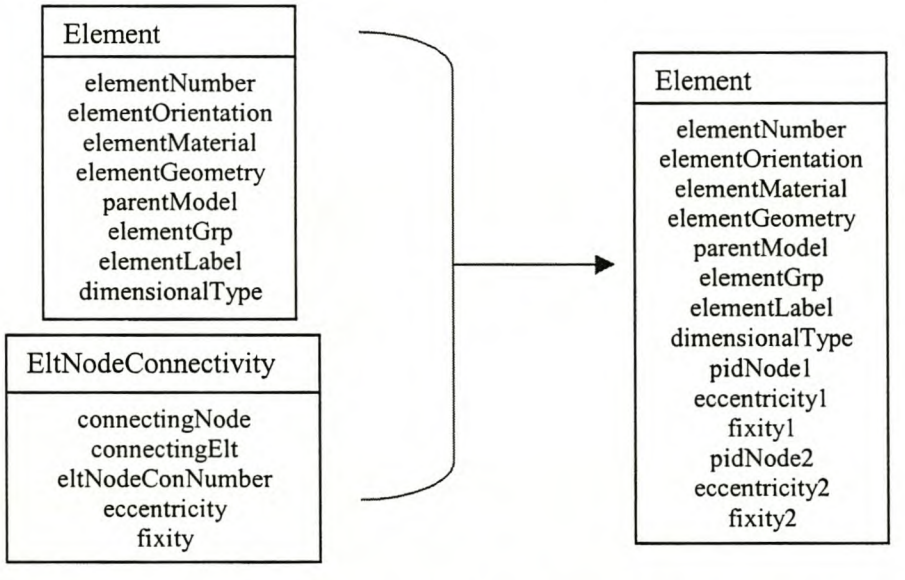


Figure 3.3: Combination of entities to form new entity Element

It is possible to combine the STEP entities *EltNodeConnectivity* and *Element* since each entry has a persistent identifier. It is also advantageous since it simplifies the model. Information is stored within

tables with the use of the persistent identifiers (pids). Information between tables are linked with the use of these pids.

Using the pids an implied relation is created rather that a prescribed relation. This means that if information within a table is changed inside the database it is unnecessary to revise the relations of the database in order to make sure that they are still valid. An example of this implied relation is demonstrated in Figure 3.4. From the seven common variables that are added to each table only the pid and version number are shown in this example.

In table *tNode* the attribute *nodeCoords* is responsible for the implied relation. By combining the pid and version number of the corresponding entry in the table *tPoint* a relation is established between the two tables.

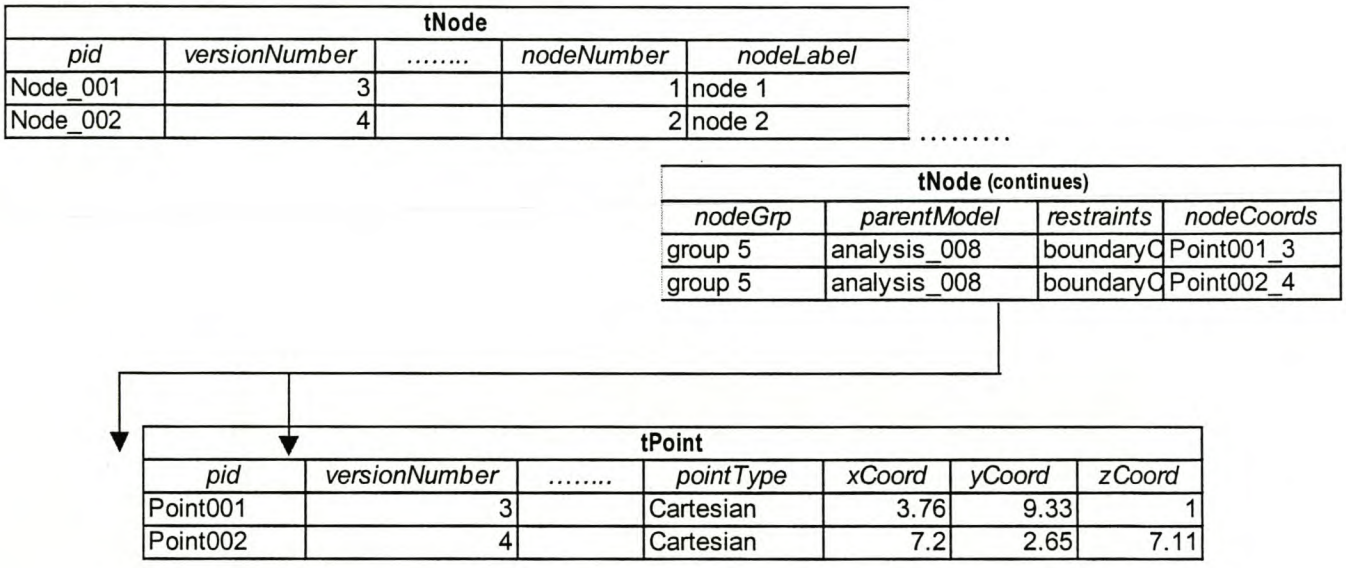


Figure 3.4: Implied relation between database tables

All the STEP entities as identified in Figure 2.25, and their corresponding database tables are listed in Table 3.3.

STEP Entity	Database PSS table
DesignAssembly	tDesignAssembly
DesignPart	TDesignPart
SPart	TSPart
SheetPart	tSheetPart
PrismaticPart	tPrismaticPart
PseudoPrismaticPart	tPseudoPrismaticPart
SheetPartGeom	tSheetPartGeom
PrismaticPartGeometry	tPrismaticPartGeometry
GeometricRepresentation	tGeometricRepresentation
Element	tElement
Node	tNode
ElementEccentricity	tElementEccentricity
Release	tRelease
Point	tPoint
BoundaryCondition	tBoundaryCondition
CoordSystem	tCoordSystem
Transformation	tTransformation
Gridline	tGridline
GridlineSet	tGridlineSet
SectionProfile	tSectionProfile
SectionProperities	tSectionProperities
AngleSect	tAngleSect
CircleSect	tCircleSect
ITypeSect	tITypeSect
RectagularSect	tRectangularSect
TTypeSect	tTTypeSect
SFeature	tSFeature
CuttingPlane	tCuttingPlane
EdgeChamfer	tEdgeChamfer
PrismaticEndFeature	tPrismaticEndFeature
ProceduralFeat	tProceduralFeat
Chamfer	tChamfer

SkewedEnd	tSkewedEnd
Notch	tNotch
Hole	tHole
Connector	tConnector
SJointSystem	tSJointSystem
BoltSystem	tBoltSystem
WeldSystem	tWeldSystem
BoltMechanism	tBoltMechanism
WeldMechanism	tWeldMechanism
Bolt	tBolt
Nut	tNut
Washer	tWasher
LocatedPart	tLocatedPart
LocatedFeature	tLocatedFeature
PartJoint	tPartJoint
JointDepFeature	tJointDepFeature
LocatedJointSystem	tLocatedJointSystem

Table 3.4: STEP entities stored as tables in database PSS

3.3 Management of Sets

Two of the identified STEP entities, listed in Table 3.3 and shown graphically in Figure 2.25, are sets, namely *DesignAssembly* and *DesignPart*. A set cannot be stored in the columns of a table since the number of elements of the set is not known beforehand and vary from case to case. The sets are consequently stored in the rows of special set tables which have predefined relations with their corresponding parent tables. A parent table is the table representing the STEP entity which is a set.

For *DesignAssembly* entities three set tables are required:

- tDesignAssemblySets
- tDesignPartSets
- tConnectorSets

For the other parent table *tDesignPart*, the set table is:

- tElementSets

The above indicates that a *DesignAssembly* is a set of *DesignAssemblies*, *DesignParts* and *Connectors*. The elements of these sets are linked to the parent table (*tDesignAssembly*) using foreign keys as shown in Figure 3.5.

It can be seen in Figure 3.5 that a design assembly can consist of other design assemblies. Out of the definition of a design assembly, described on section 2.5 (also see Appendix A.1) and shown graphically in Figure 2.24, it is known that a design assembly is a set of structural parts and structural joint systems. This means that a simple industrial steel structure consists of a single design assembly comprising of any number of sub-assemblies.

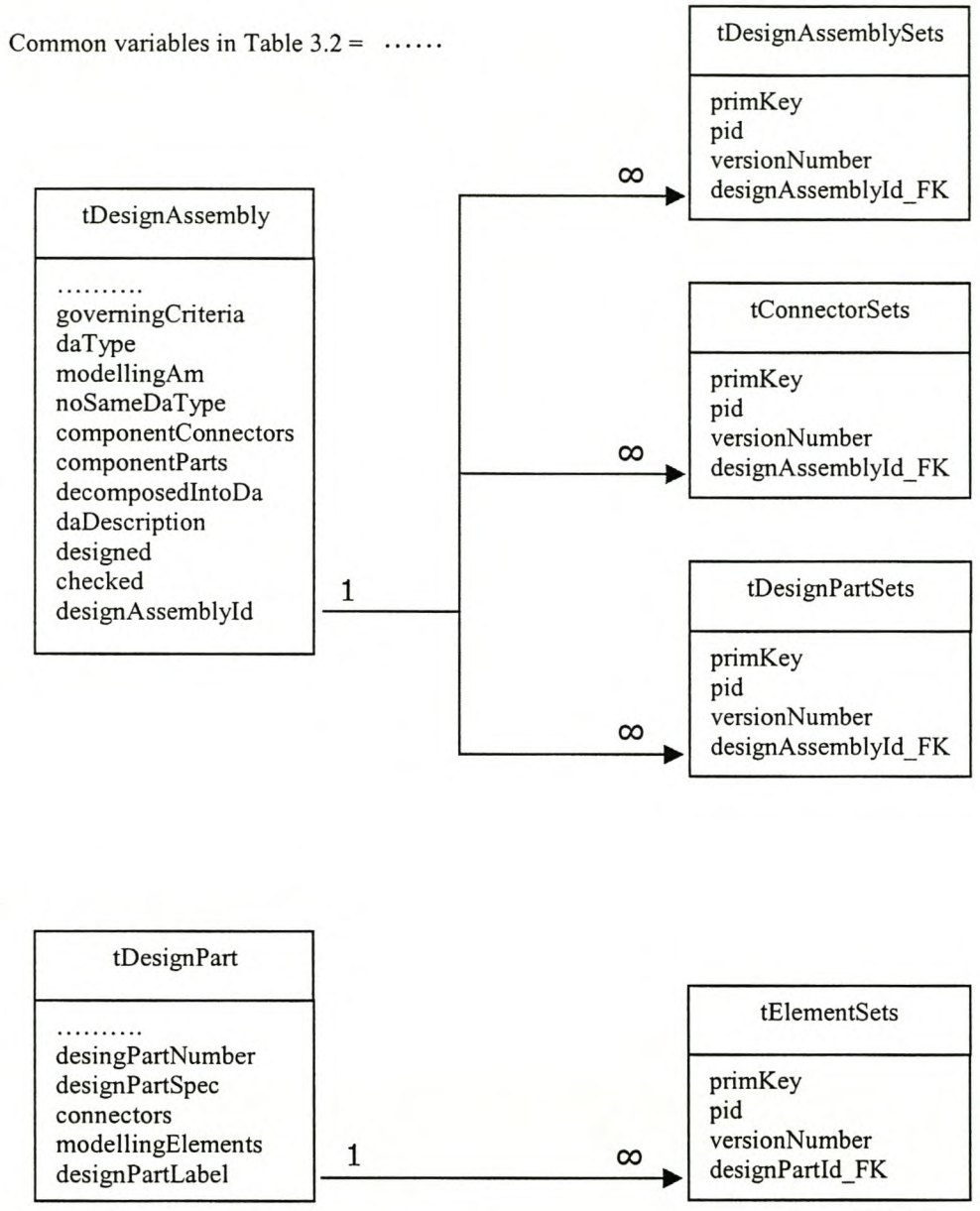


Figure 3.5: Relations between the different database tables

Consider for example the storing of several design parts of a design assembly in table *tDesignParts*. Design parts are a representation of a physical structural part for member design purposes. The layout of storing information of such a part is shown in Figure 3.5.

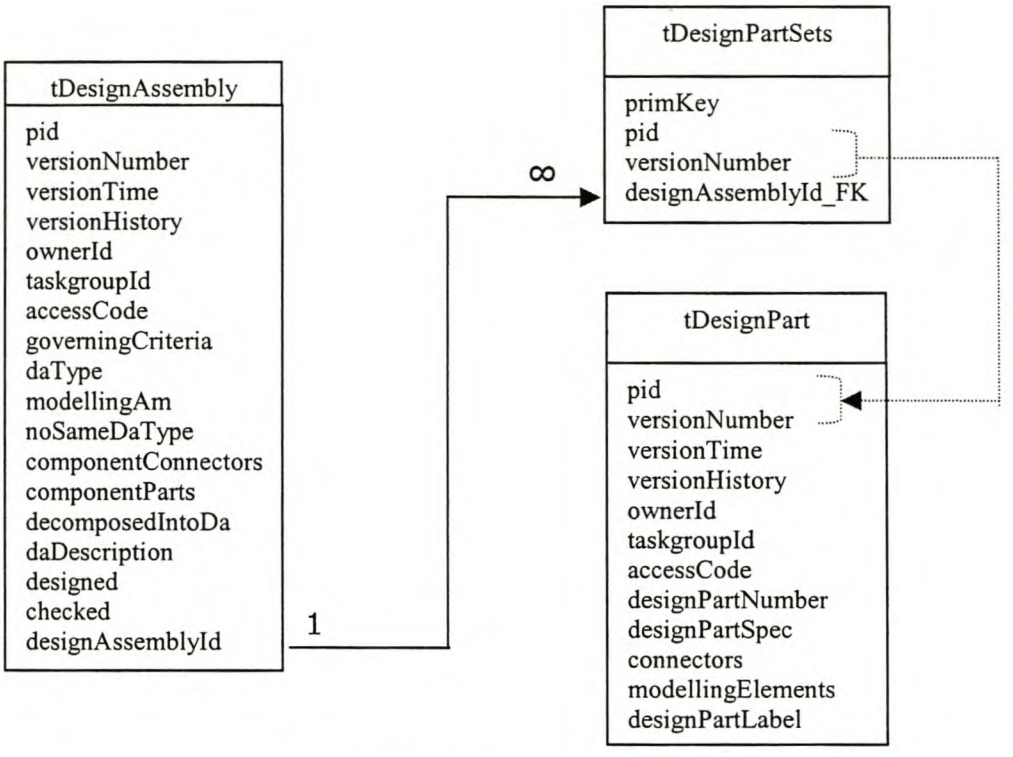


Figure 3.6: Storing sets of information data

In table *tDesignAssembly* there is an additional attribute added to allow the connection between the different set tables. This attribute is *designAssemblyId*, and has the value of an automatically generated number. Figure 3.6 demonstrates how the internal relation between tables is constructed: The relation is established using the attribute *designAssemblyId*. The value of this attribute is duplicated in the *tDesignPartSets* table storing it, as a foreign key, in the attribute *designAssemblyId_FK*. This means that if the value for a *designAssemblyId_FK* in table *tDesignPartSets* is the same as the value *designAssemblyId* in table *tDesignAssembly* that that design part forms part of the design assembly under review.

The relation is a one-to-many relation, which means that a single design assembly can consist of several design parts. The attributes of the set table *tDesignAssemblySets* are described in Table 3.4.

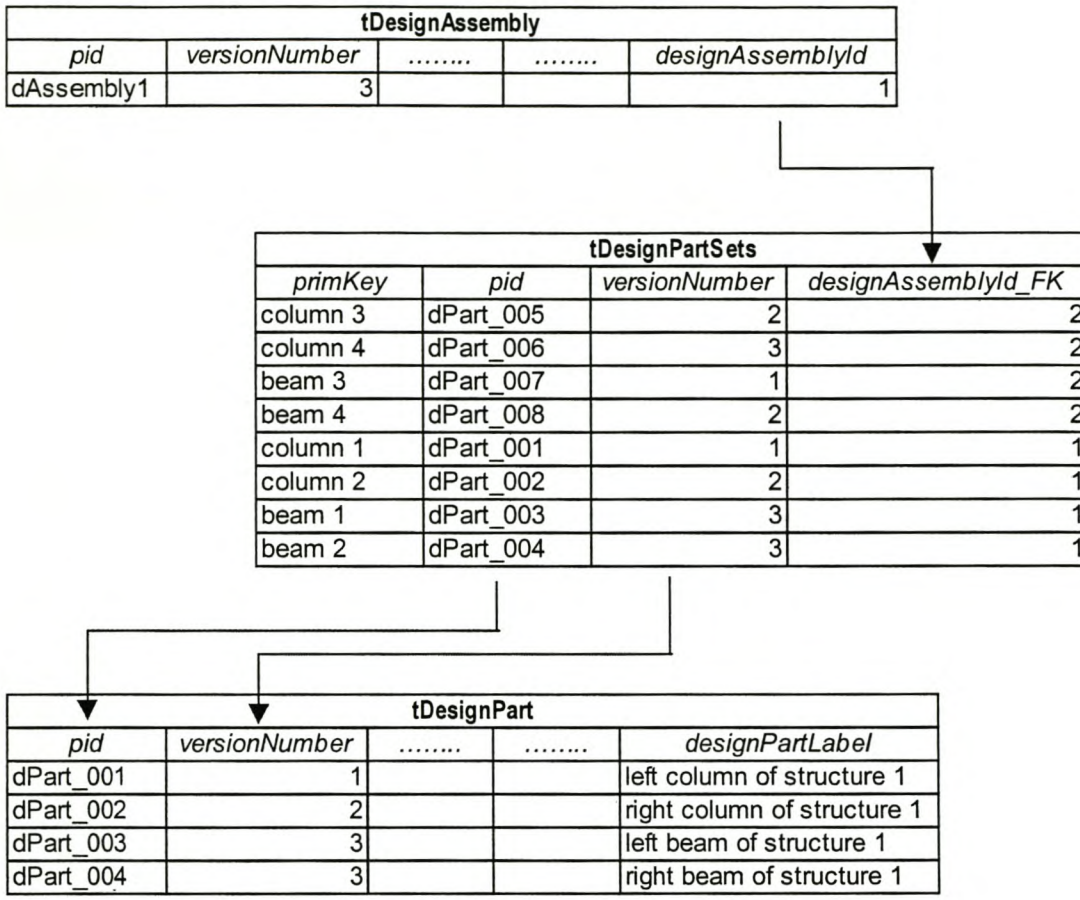
Attributes	Description
primKey	Primary key of set table
pid	Persistent identifier of a DesignPart which is an element of this DesignAssembly
versionNumber	Version number of DesignPart
designAssemblyId_FK	Foreign key identifies 'this' DesignAssembly

Table 3.5: Definition of tDesignPartSets attributes

The pid and version number in *tDesignPartSets* are the same as the pid and version number in table *tDesignpart*. Therefore once the pid and version number have been retrieved from the sets table they can be used to find the corresponding design part in table *tDesignPart*, shown in Figure 3.7.

In Figure 3.7, a design assembly with pid *dAssembly1* and version number 3, has a *designAssemblyId* of 1 which is allocated automatically. The corresponding foreign key, *designAssemblyId_FK*, in the sets-table is allocated the same number as *designAssemblyId*, in this case also a 1.

This means that all the pids and version numbers in the sets-table corresponding to a foreign key value of 1 belong to the design assembly *dAssembly1*. These pids and version numbers are used to select the appropriate design parts from the design part table (*tDesignPart*). From these identified design parts other information can be gathered and used in the application of the exchanging data with the database.



tDesignAssembly

pid	*versionNumber*	.....	.....	*designAssemblyId*
dAssembly1	3			1

tDesignPartSets

primKey	*pid*	*versionNumber*	*designAssemblyId_FK*
column 3	dPart_005	2	2
column 4	dPart_006	3	2
beam 3	dPart_007	1	2
beam 4	dPart_008	2	2
column 1	dPart_001	1	1
column 2	dPart_002	2	1
beam 1	dPart_003	3	1
beam 2	dPart_004	3	1

tDesignPart

pid	*versionNumber*	.....	.....	*designPartLabel*
dPart_001	1			left column of structure 1
dPart_002	2			right column of structure 1
dPart_003	3			left beam of structure 1
dPart_004	3			right beam of structure 1

Figure 3.7: Storing several Design Parts

The database schema presented above is extremely simple and consequently flexible. Most of the relations between entities are established with the aid of persistent identifiers and do not rely on predefined relations within the database. The exception is the set tables, which rely on predefined relations, but the probability that these could cause a serious problem is very low.

CHAPTER 4

STRUCTURAL CONCEPT DEFINITION

4.1 Background:

The entities required to describe a simple industrial steel frame, and the database model for persistent storage of these entities were described in the previous chapters. In this chapter a simple application supporting the definition of the structural concept of a steel frame is described. This application, called CadDrawing, was developed in order to demonstrate the use of the STEP entities and the database model.

Application CadDrawing allows for the description of the structural concept of a simple industrial structure, the storage of the defined structure as a product model, and the retrieval and graphical display of a structure that has been stored as a product model before.

Creating the runtime structure proved a difficult task since the STEP documents are not intended for direct runtime implementation. CadDrawing was designed in such a way that it can be expanded stepwise to include the various aspects of the STEP protocol.

4.2 Specification of the Graphical User Interface

4.2.1 Graphical Components:

Java permits the use of predefined graphical components as well as programmer-defined graphical components in the user interface. The predefined components (frames, panels, dialog boxes, labels, buttons and text fields) are suitable for activity control and alphanumerical presentation. For the geometric processing aspects of the interface, it is necessary to develop programmer-defined components. The two types of components are integrated in a common frame.

4.2.2 Frame Layout:

When CadDrawing is called from the operation system shell, a frame appears on screen with the title *"Gui : CAD drawing"*. There are 4 panels inside the frame: the drawing panel, icon panel, control panel and graph panel as shown in Figure 4.1.

The graph panel is situated in the center of the frame, the drawing panel in the north, the icon panel on the left and the control panel at the bottom. The graph panel is used to draw the graphical components. It has scroll bars to enable the user to view drawings that are too large to be shown completely in the frame.

The drawing panel contains three radio button groups, the component group, the selection group and the mode group. Only one button of each radio button group can be selected at a given time. These groups are orange in colour. The component group consists of the buttons *"Node"*, *"Edge"* and *"Member"*. These buttons are used to specify the type of input to which the subsequent input operations apply. The selected component remains valid for all subsequent input operations until changed to another component of the group.

The selection group consists of the buttons *"Nodes (S/G)"*, *"Edges (S/G)"*, *"Members (S/G)"* and *"S/G All"*. These buttons are used to specify the type of component that will be stored to or returned from a database. They are used in conjunction with the "Store Data" or "Get Data" buttons in the icon panel to either store data to a database or to get data from a database.

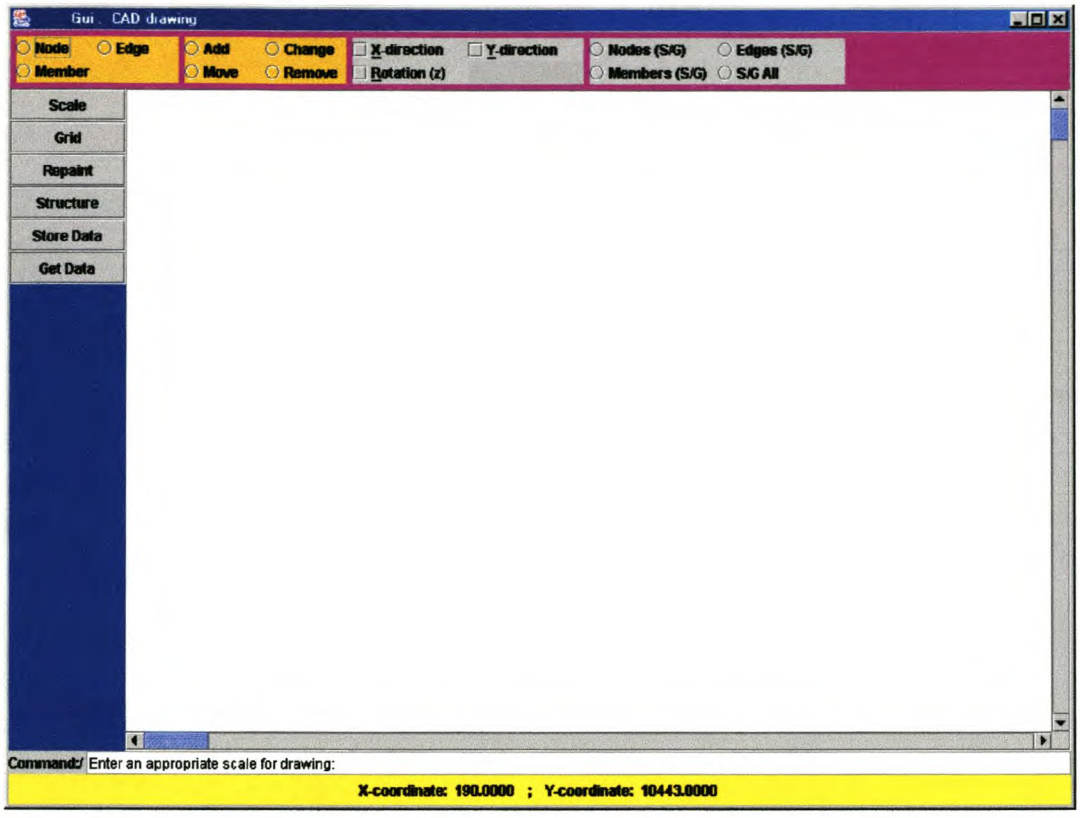


Figure 4.1: CadDrawing when it is loaded from the shell and the layout of the 4 panels

The mode group consists of the buttons *"Add"*, *"Change"*, *"Move"* and *"Remove"*. It is used to specify the mode of the graphical input. The semantics of the mouse click in the graph panel depend on the input mode.

The icon panel contains the buttons *"Scale"*, *"Grid"*, *Repaint"*, *"Structure"*, *"Store Data"*, *"Get Data"*. These buttons are grey in colour. Their function will be described in the following sections.

The control panel of the frame consists of two zones, the command zone and the coordinate zone. The command zone guides the user through the steps of the drawing process. At the beginning of the dialog the user is prompted to select a scale. After the scale is specified, further prompts appear in this command line. The coordinate zone shows the user coordinates of the mouse pointer while it is within the graph panel.

4.2.3 Coordinate System:

In order to make the interface suitable for engineering applications, the standard left-handed Java coordinate system with origin at the top left corner of the graphical component, x-axis running from left to right and y-axis running from top to bottom is replaced by a conventional right-handed coordinate system with origin at the bottom left of the graphical component, x-axis running from left to right and y-axis running from bottom to top. User coordinates map unto the pixel coordinates used in Java.

Before starting the frame presentation the user must enter a desired scale to enable the system to set the user coordinates. Selecting the *"Scale"* button within the icon panel opens a dialog box where the desired scale is entered.

A scale can also be changed, again clicking on the *"Scale"* button and inputting a new value. If one does not wish to implement the selected scale, one is able to cancel this action by clicking on the *"Cancel"* button inside the scale dialog box.

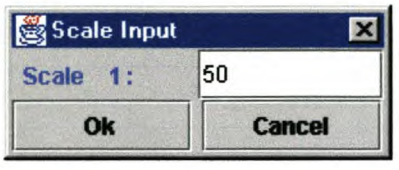


Figure 4.2: A scale of 1:50 has been entered.

The implemented conventional right-handed user coordinate system is used while the user draws any graphical image. The origin is shifted to a new position. It is relocated to a position with pixel coordinates of (30; 30). This is done to allow the user to be able to see the coordinates drawn next to the gridlines. This new origin coincides with the origin of the coordinate grid.

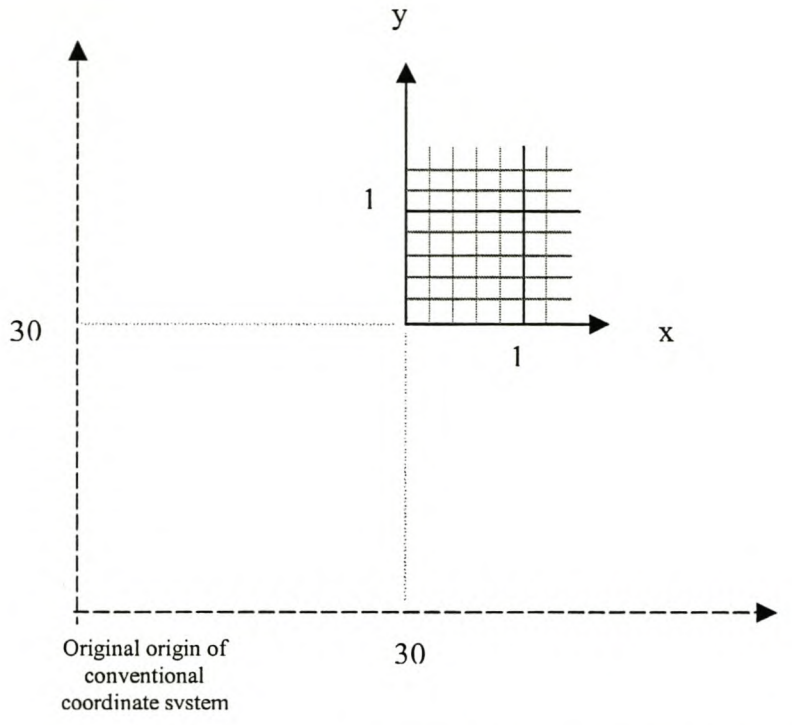


Figure 4.3: Transformation of the origin:

4.2.4 Coordinate Grid:

A grid can be drawn on the graph panel to assist the user in positioning the mouse. The coordinate grid consists of fine gridlines and of main gridlines, which cover every 5th fine gridline.

The main grid lines are marked with their user coordinates. For a chosen scale of 1:50 and lower, coordinates are shown at each main gridline. For a scale greater that 1:50, coordinates are shown at every 2nd main gridline.

A grid is drawn once a scale has been entered, and the *"Ok"* button pressed inside the scale dialog box. The grid is also updated if the scale is changed. One can remove the grid by clicking on the *"Grid"* button inside the icon panel.

4.2.5 Reference Frame:

The geometry of the steel structure is described by positioning the structural members relative to a user-defined reference frame. The reference frame is a directed graph consisting of nodes and edges.

4.2.6 Node:

A node is described by its user coordinates. It is shown on the graph panel as a hollow square, red in colour. Each edge of the reference frame starts and ends at a node. Each support of the structure is specified relative to a node of the structure.

Selecting the *"Node"* button and one of the modes allows the user to do the following:

- In the add mode a node is drawn wherever the mouse pointer is clicked.
- In the move mode the user can move a node to a different location by clicking on the node and dragging it to its new location.
- In the remove mode, the user removes a selected node by clicking on the node. If an edge is in incident at the removed node, it will be removed together with the node.
- In the change mode, a dialog box appears with the options of changing the label for the chosen node. One is also able to specify the exact coordinates for the node by inputting them in the appropriate text fields. This makes it possible to place the node with great accuracy. At the top of this editing dialog box are three buttons. These are used to specify the prescribed support degrees of freedom at a specific node for the x- and y-direction as well as the rotation of the node. If a button is selected to indicate a support the colour of the button will change to yellow as shown in Figure 4.4 below. In Figure 4.4 this means that the node is locked in the x- and y-direction as well as being locked against rotation.

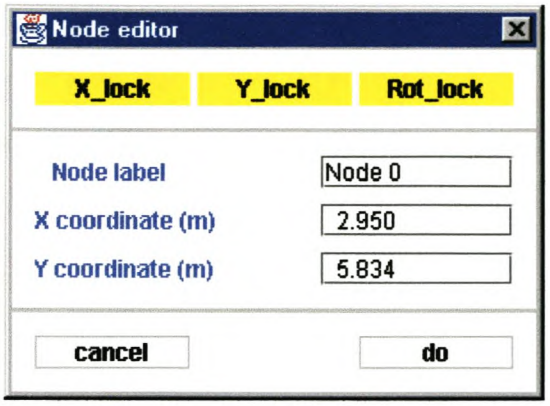


Figure 4.4: Editing dialog box for a node

4.2.7 **Edge**:

Edges of the reference frame are the reference lines for the positioning of members of the structure. The location of the member is specified with respect to end points and the direction of the reference line. If the member does not have an eccentricity its centreline coincides with the reference line.

Edges are drawn in a blue colour while the members are drawn in a grey colour. An arrow is added to the centre of the edge to show its direction. The direction of the edge is determined by the start and end node of that edge.

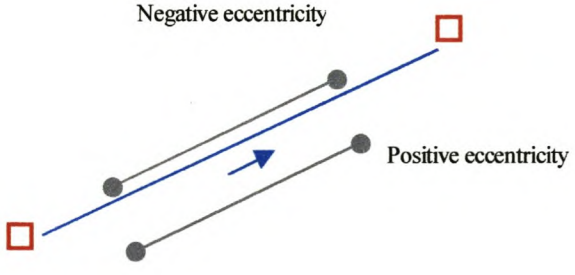


Figure 4.5: Sign of the eccentricity of a member.

The arrow always points towards the end node. The direction of the edge is used to define positive and negative eccentricity: The eccentricity of a member is positive if the centreline of the member lies to the right of the edge, looking in the direction of the arrow.

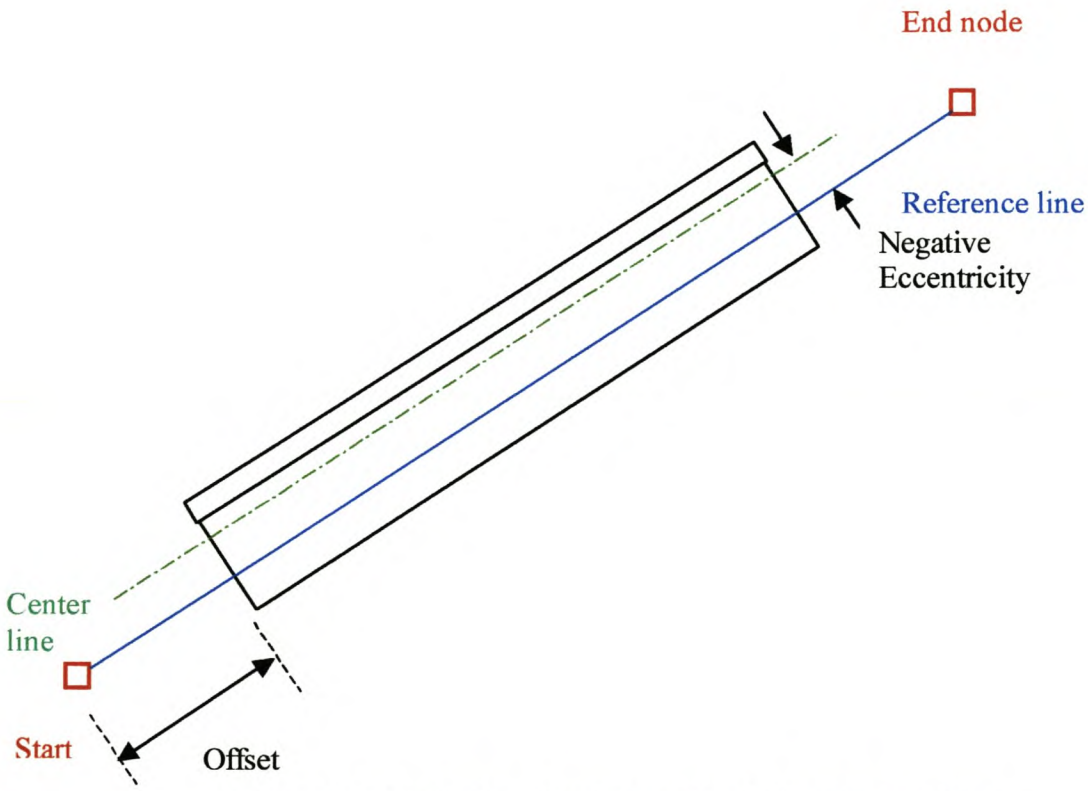


Figure 4.6: Positioning of a steel member in the reference frame.

Selecting the *"Edge"* button and one of the modes allows the user to do the following:

- In the add mode the start node is selected first. This changes the colour of the node. It now has a black border and is filled with a light grey colour. If the end node is then selected, an edge is drawn between the selected nodes. The persistent identifiers of the nodes are stored. With the persistent identifiers an edge is drawn and stored in an element hash set.

The persistent identifiers of the nodes enable the edge to receive its starting x- and y-coordinates from the start node, as well as its end x- and y-coordinates from the end node. Having the start and end coordinates an edge is drawn. If a node is removed and an edge is linked to it, the edge is unable to receive the x- and y-coordinates of the removed node. No edge can be drawn. An arrow pointing to the end node indicates the direction of the edge.

- The change mode is not used for edges.

- If an edge is selected in the remove mode, it is removed from the reference frame. The end nodes of the edge are not affected.

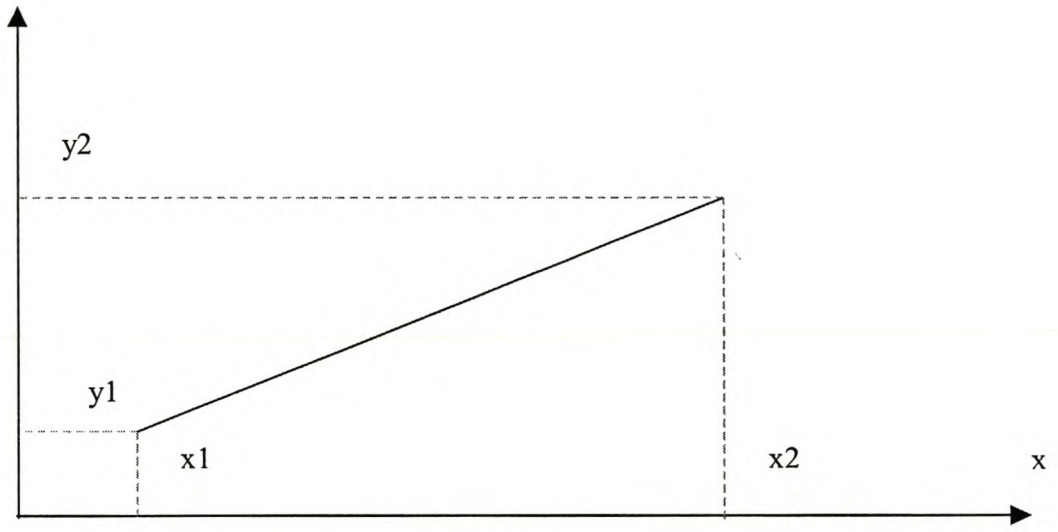


Figure 4.7: Drawing an edge with start and end coordinates

- An edge is moved by moving its end nodes. Once a node is moved, any element connected to that node moves with it while the other end of the edge stays at its original position. The edge is shifted to the moved nodes. This is illustrated in Figure 4.7.

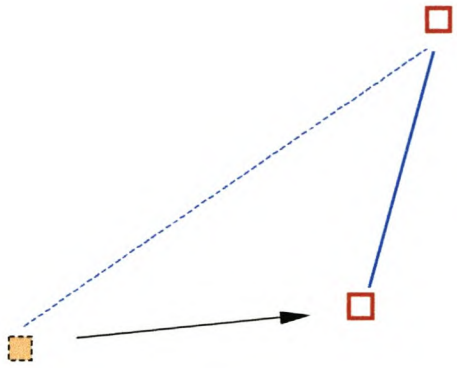


Figure 4.8: Reorientation of an edge

4.2.8 Member:

Members are drawn in a grey colour. Members are represented in the reference frame by edges. An edge represents a reference line of a member. This can be seen from Figure 4.5. The user can edit the member input by selecting the *"Member"* button in the component radio group together with the *"Change"* button in the mode group.

In the change mode, a dialog box will appear with the option of changing the label of the member. The members are labelled "Member i", where i = 0,1,2,3,…

The user can also specify an eccentricity for the element with respect to an edge. As a default the eccentricity is set to zero. A zero eccentricity means that the reference line, represented by the edge, coincides with the centreline of the member. The user can also input geometrical values for an I-type profile as is shown in Figure 4.9.

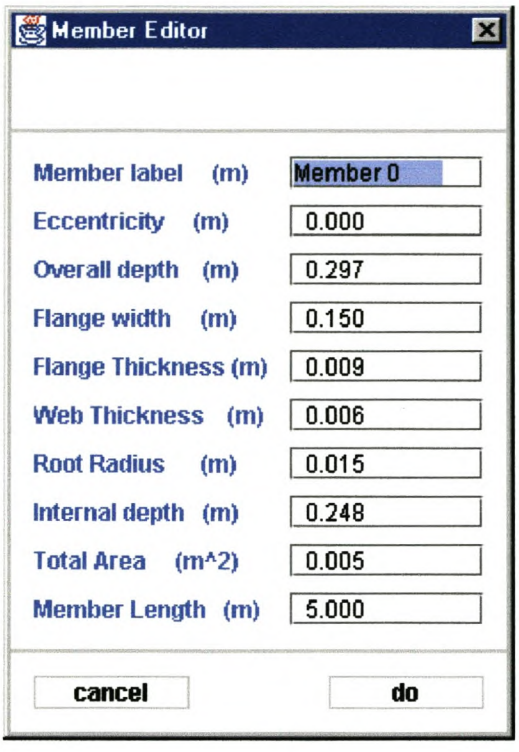


Figure 4.9: Editing dialog box for a member

Selecting the *"do"* button will store the entered values while the *"cancel"* button ignores all input. The entered values for an I-profile make it possible to continue the definition of the reference frame. The data defined in Figure 4.9 is shown graphically in Figure 4.10.

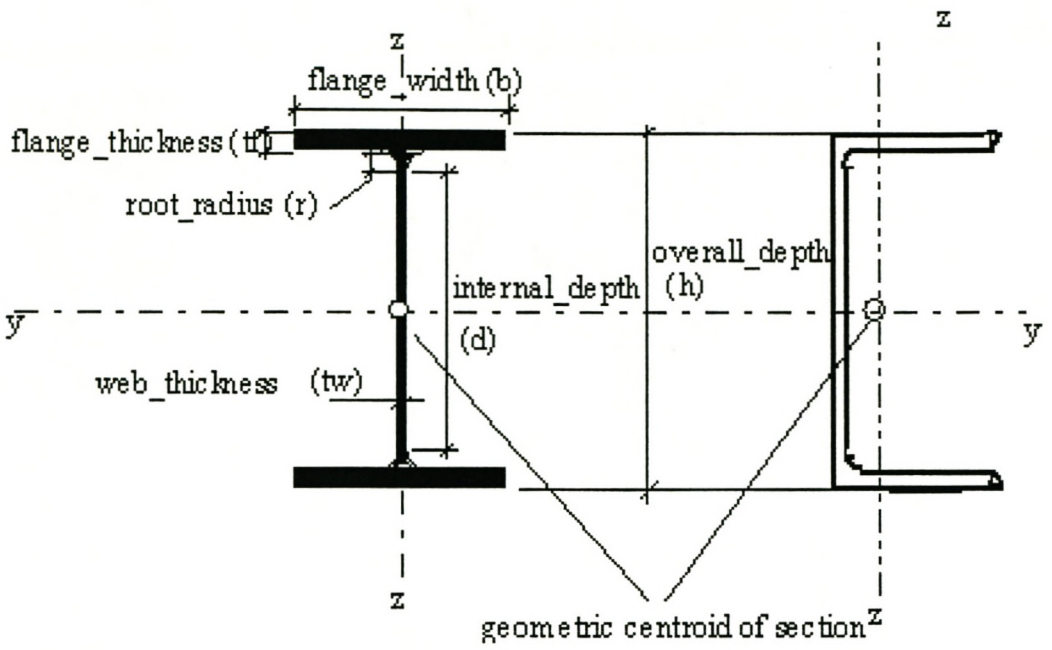


Figure 4.10: Description of an I-profile

4.3 Implementation of the Graphical User Interface (GUI)

A class diagram of the GUI is shown in Figure 4.11. The different classes are described in the following sections.

4.3.1 CadDisplay:

This class handles the construction of the entire frame. The different panels are set out. Buttons are grouped together as either ordinary buttons, radio buttons or checkbox buttons.

The component buttons and the mode buttons are also allocated values. These values are used to determine which selection of component button together with a mode button is currently active. With this information different operations are carried out. The values and attributes associated with the different component buttons and mode buttons are as follows:

```
final public static int NODE      = 10;
final public static int EDGE      = 20;
final public static int MEMBER    = 30;
static                 int component;

final public static int ADD       = 1;
final public static int CHANGE    = 2;
final public static int REMOVE    = 3;
final public static int MOVE      = 4;
static                 int mode;

final public static int XDIRECT   = 15;
final public static int YDIRECT   = 16;
final public static int ROTATE    = 17;
static                 int support;
```

Figure 4.11: Class Diagram of Graphical User Interface

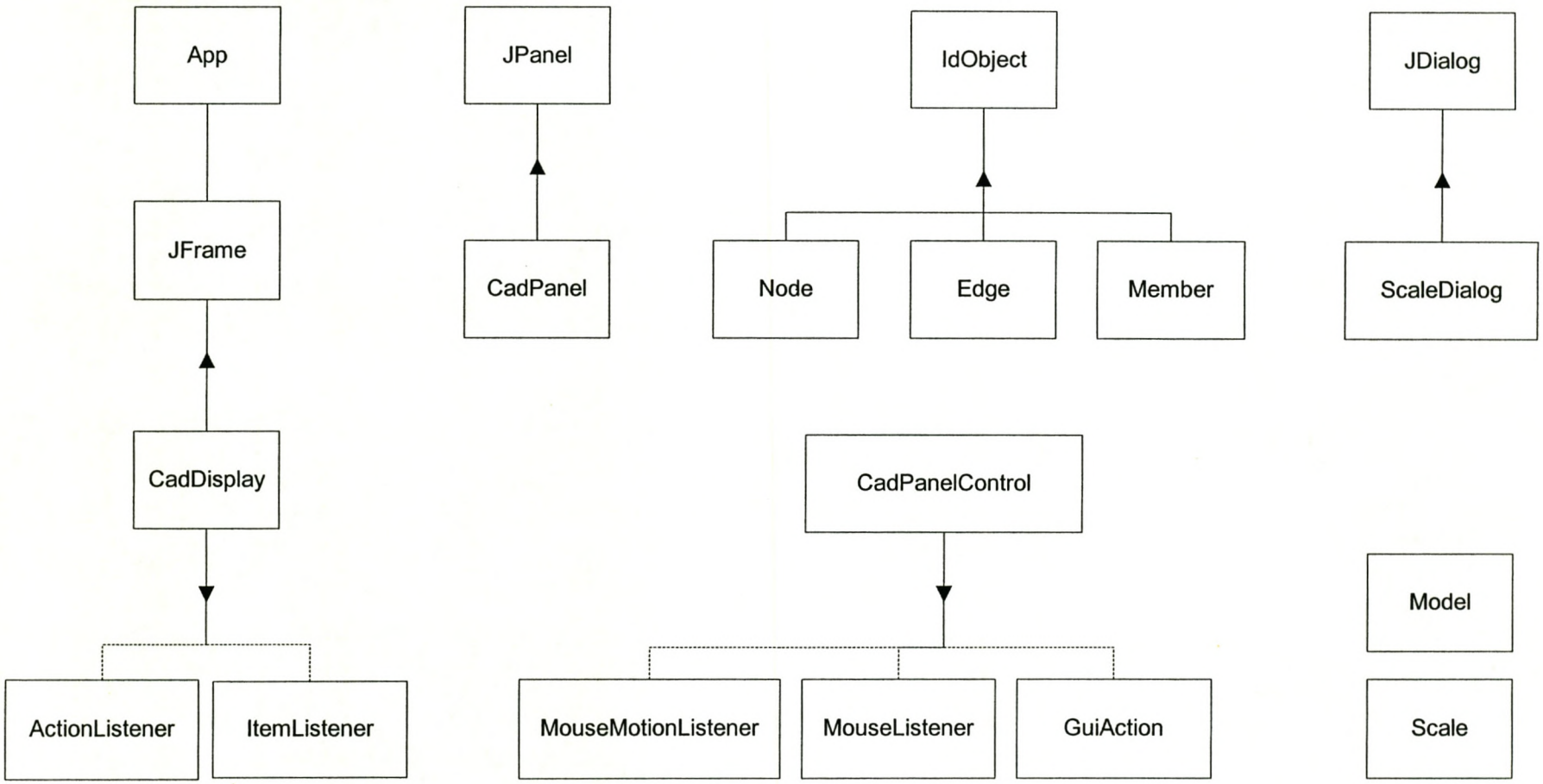


Figure 4.11: Class Diagram

The construction and layout of the different panels and buttons is done with the following methods.

controlPanel():	Constructs the control panel containing the command zone and the coordinate zone.
iconPanel():	Defines the icon panel containing the buttons "Scale", "Grid", "Repaint", "Structure", "Store Data" and "Get Data".
drawingPanel():	Constructs the drawing panel containing the two radio groups , the component group and the mode group.
cadDisplayLayout():	Specifies the layout of all the panels within the frame.
buttons():	Creates all the buttons that are used during the construction process.

4.3.2 CadPanel:

This class is responsible for all the graphics that is drawn in the graph panel. When this class is called from the shell it draws the axis for the conventional coordinate system. When a scale is entered, methods within this class calculate the scaling factor for the user coordinates. This scaling factor is used to scale the pixel coordinates to real world user coordinates. The scaling factor for the user coordinates is called *"oneMeter"*. It is calculated using the number of pixels per millimeter for the computer and the scale input.

The methods for drawing the grid are also constructed within this class. If the "Grid" button is selected from the icon panel the system performs a check to see if the grid is active or whether it is inactive. An active grid appears on the screen. If the grid was active it is automatically set to inactive, and the grid lines are removed from the graphical context. If the initial value for the grid showed that it was inactive it will be changed to active and the grid is drawn on the graphical context. The grid variables are shown with their corresponding values.

```
//GRID VARIABLES-------------------------------------------------------------
final static int BORDER          = 30;
final static Color fineGridColor    = new Color(225, 225, 225);
```

```
final static Color mainGridColor  = new Color(200, 200, 200);
final static Color borderColor    = Color.black;
final static Color numberColor    = new Color(100, 100, 100);
boolean            isGrid         = false;      //initial value

//SCALE VARIABLES-------------------------------------------------------------
double pixPerMM;
double enteredScale;
static double oneMeter            = 1.0;        //initial value
```

The most important methods in this class are shown below. These methods are responsible for the graphical representation, setting out the axis for the user coordinates and drawing the grid.

paintComponent(Graphics g): Draws all graphics that are shown on the graph panel.

getPathXArrow(): Draws an arrow in the x-direction indicating the user coordinates.

getPathYArrow(): Draws an arrow in the y-direction indicating the user coordinates.

drawGrid(Graphics g): Draws the coordinate grid.

isGrid(): This method checks to see if the grid is active or inactive.

setGrid(boolean b): Sets a boolean value for the grid. A true value indicates that the grid is active.

public double scaleCalc(): Calculates the scaling factor *"oneMeter"*.

4.3.3 CadPanelControl:

This class is responsible for all mouse operations done within the graphical context. Dependent on the selected modes and the type of mouse operation that is performed graphical components can be added, moved, changed and removed from the graphical context. The most important methods are described below.

mouseClicked(MouseEvent e): With the left mouse button this method adds and removes nodes, edges and members.
With the right mouse button this method edits information of nodes and members.

mousePressed(MouseEvent e): This method is used to select a node before it is moved.

mouseDragged(MouseEvent e): Moves a node to a new position.

mouseReleased(MouseEvent e): Returns the coordinates of the node's new position.

mouseMoved(MouseEvent e): Receives the coordinates for the mouse pointer in order to update the user coordinates shown in the coordinate zone.

getXCoord(MouseEvent e): Returns the x-coordinate when an operation is performed

getYCoord(MouseEvent e): Returns the y-coordinate when an operation is performed

4.3.4 Scale:

This class receives the screen size of the computer. The number of pixels per inch is received and is converted to millimeters. This converted value is the number of pixels the computer screen displays per millimeter.

getPixPerMM(): This method returns the real number of pixels per millimeter.

4.3.5 ScaleDialog:

If the "Scale" button in the icon panel is pressed this class is called by the system. It constructs a dialog box that prompts the user for a scale input. The user can, after inputting a scale, continue or cancel this operation. The input is first read as a string and is then converted to an integer value.

getLabelText(): This method receives the entered scale which is a String and casts it to an integer value.

4.3.6 Model:

Contains all the hash set for the different components. The hash sets stored are the node set, the edge set and the member set.

4.3.7 IdObject:

This class is used to create a persistent identifier for each of the components. Each component is then stored in a hash map as ordered pairs. The map is used primarily to determine the reference of an object with a known identifier.

4.3.8 Node:

This class is called each time a new node is created. It contains all operations that deal with a node. The node is described by its user coordinates. These coordinates and the node label is edited inside the edit dialog box. A node also has a persistent identifier.

paintNode(Graphics g): Paints the node and calls other methods needed to draw the supports of the node.

checkCoordinates(double xClick, double yClick): This method is used to check if the coordinates received from a mouse click correspond to the coordinates of a node. If the clicked coordinates are the same as the coordinates of a node, a true value is returned.

Checked(String idGiven): Checks to see if a given identifier is the same as the identifier of the node. A true value is returned if the identifiers are the same.

xDirection(Graphics g): Draws a line indicating that the node is locked in the x-direction.

yDirection(Graphics g): Draws a line indicating that the node is locked in the y-direction.

rotation(Graphics g): Draws a filled square around the node to show that it is locked against rotation about the z-axis.

4.3.9 Edge:

This class is called each time a new edge is created. It contains all operations that deal with an edge. An edge contains a persistent identifier that makes it unique. With this it is possible to get an edge from the edge hash set.

drawEdge(Graphics g): Draws an edge which is the reference line.

drawDirection(Graphics g,
Node n1,
Node n2): This method is used to indicate the direction of the edge. The direction is used to specify the eccentricity of a member.

checkEdgeCoordinates(int xClick,
int yClick): This method is used to check if the coordinates received from a mouse click correspond to the coordinates of an edge. If the clicked coordinates are the same as the coordinates of the edge, a true value is returned.

4.3.10 Member

This class is called each time a new member is created. Members are created with an edge. This class contains all operations that deal with a member. A member also has a persistent identifier. All values that van be edited within the edit dialog box are shown beneath. These values all correspond to a I-type section.

drawMembers(Graphics g): Draws a representation of the members using the reference frame. This also shows the eccentricity of a member's centerline relative to the reference line.

checkMemberCoordinates(int xClick, int yClick): This method is used to check if the coordinates received from a mouse click correspond to the coordinates of a member. If the clicked coordinates are the same as the coordinates of the member, a true value is returned.

4.4 Sample Application GUI Screen Shots

Figure 4.12: Shows the frame that is constructed when CadDrawing is called from the shell.

Figure 4.13: Shows the coordinate grid drawn for a scale input of 1:100.

Figure 4.14: Shows the coordinate grid drawn for a scale input of 1:50.

Figure 4.15: Shows the adding of nodes. The prescribed support degrees of freedom for a node are also shown. The nodes are represented as filled red squares and not hollow red squares as described in the previous sections. This is done to make it easier for the reader to view the nodes.

Figure 4.16: Shows how edges are added to the reference frame. The change in color of a selected node is also shown.

Figure 4.17: Shows the members of the structure. The member centrelines coincide with the reference lines because there is no eccentricity prescribed for the members.

Figure 4.18: The members are assigned an eccentricity. The columns are assigned a positive eccentricity and the beams a negative eccentricity.

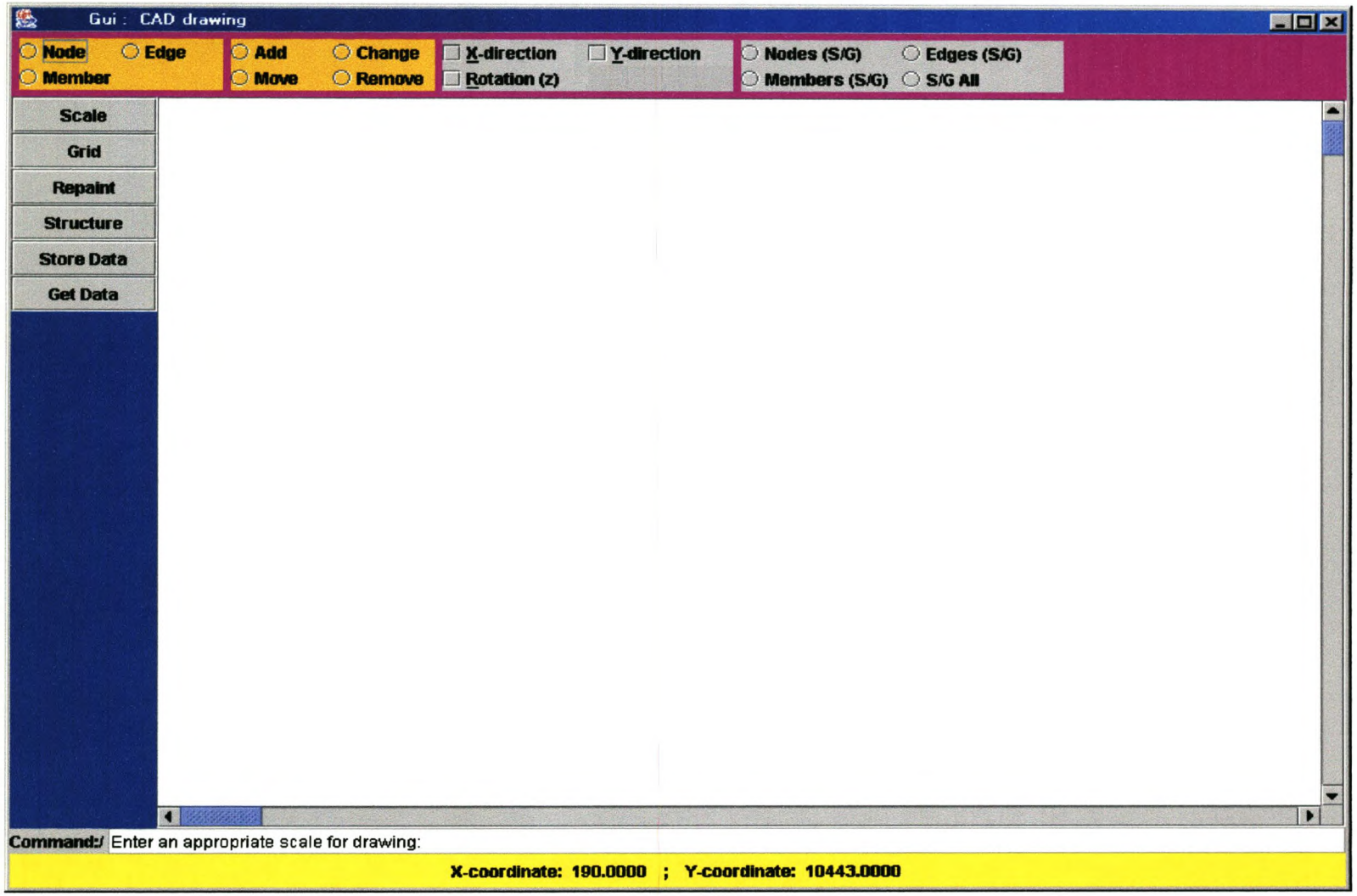


Figure 4.12

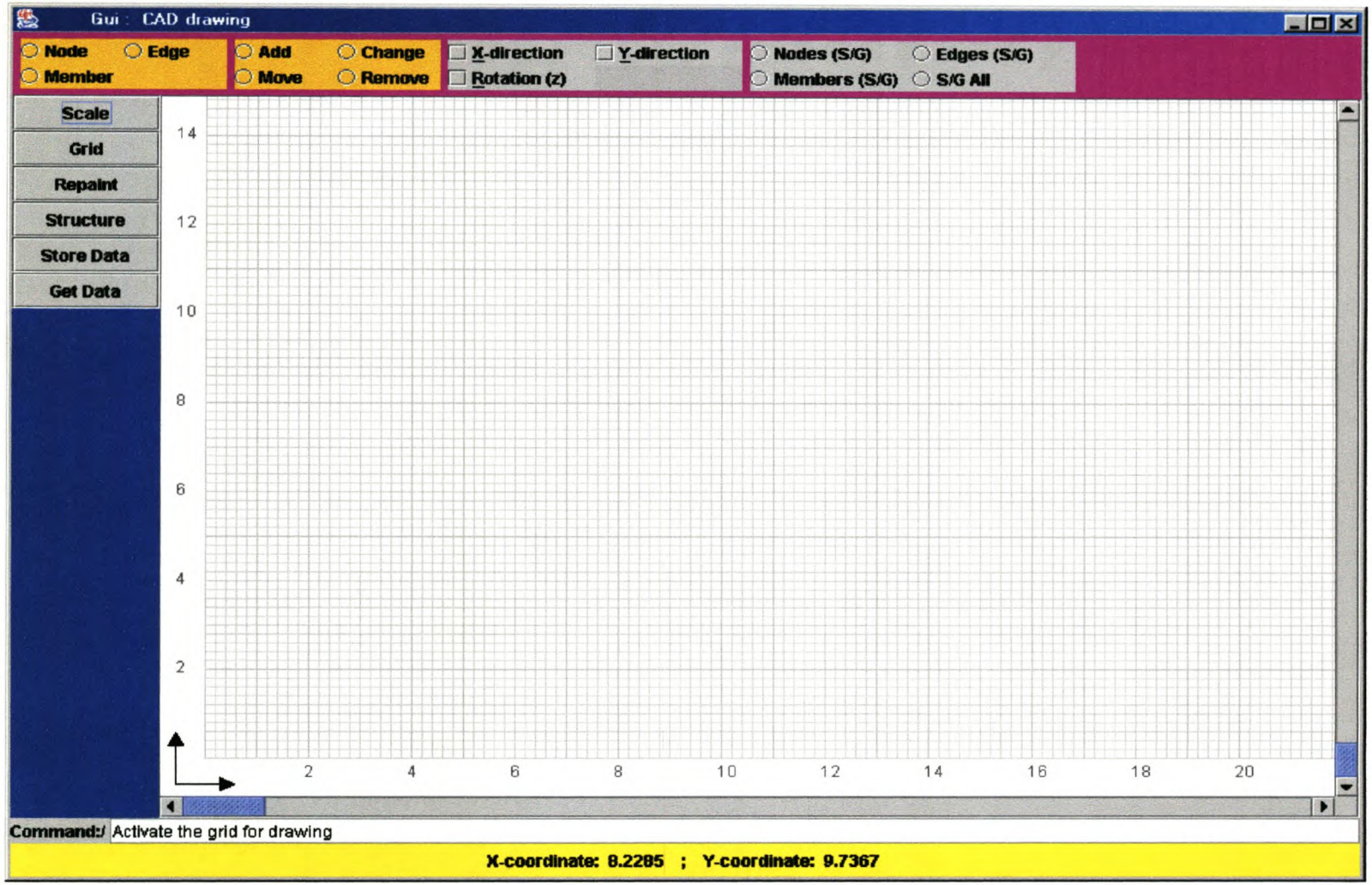


Figure 4.13

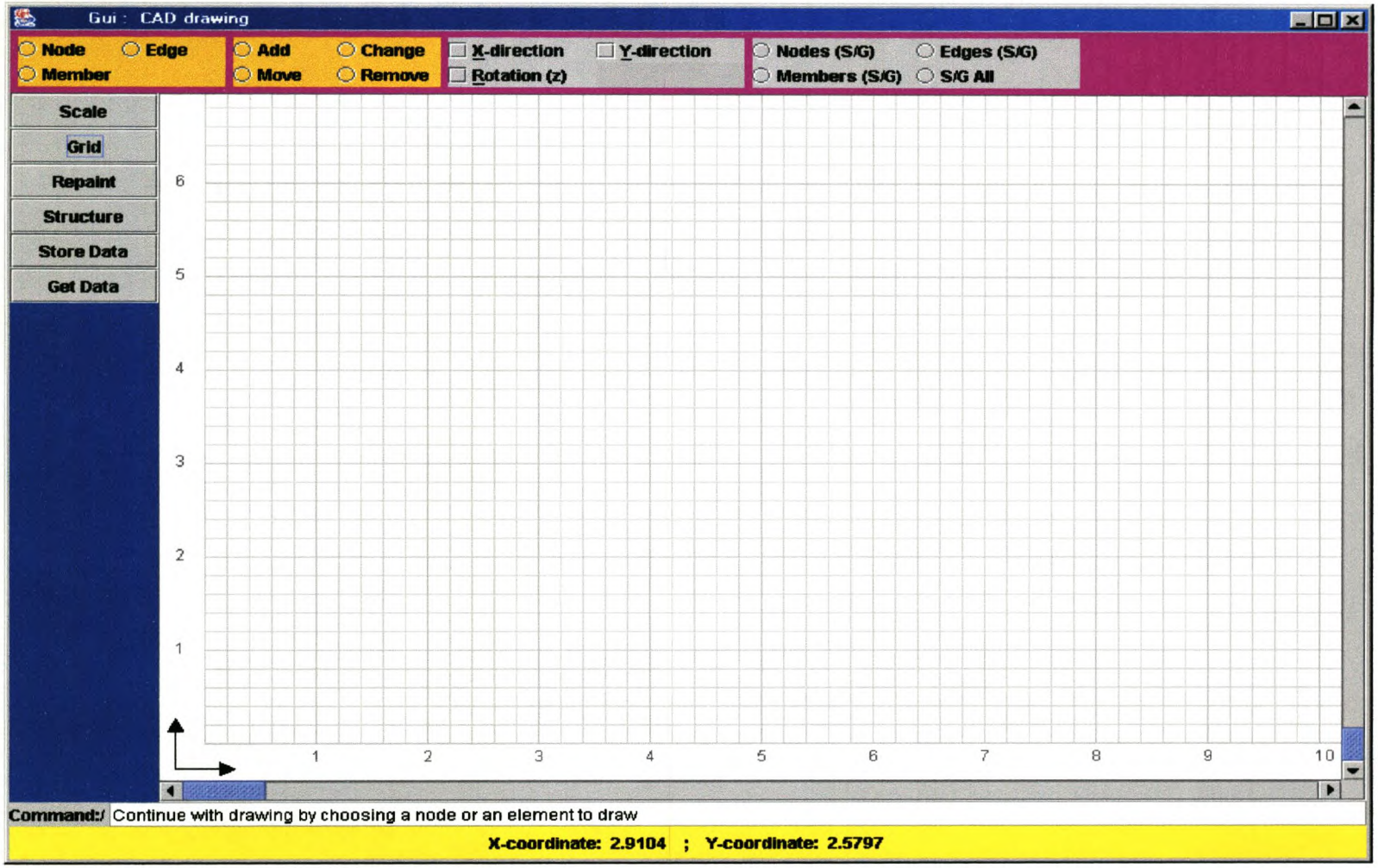


Figure 4.14

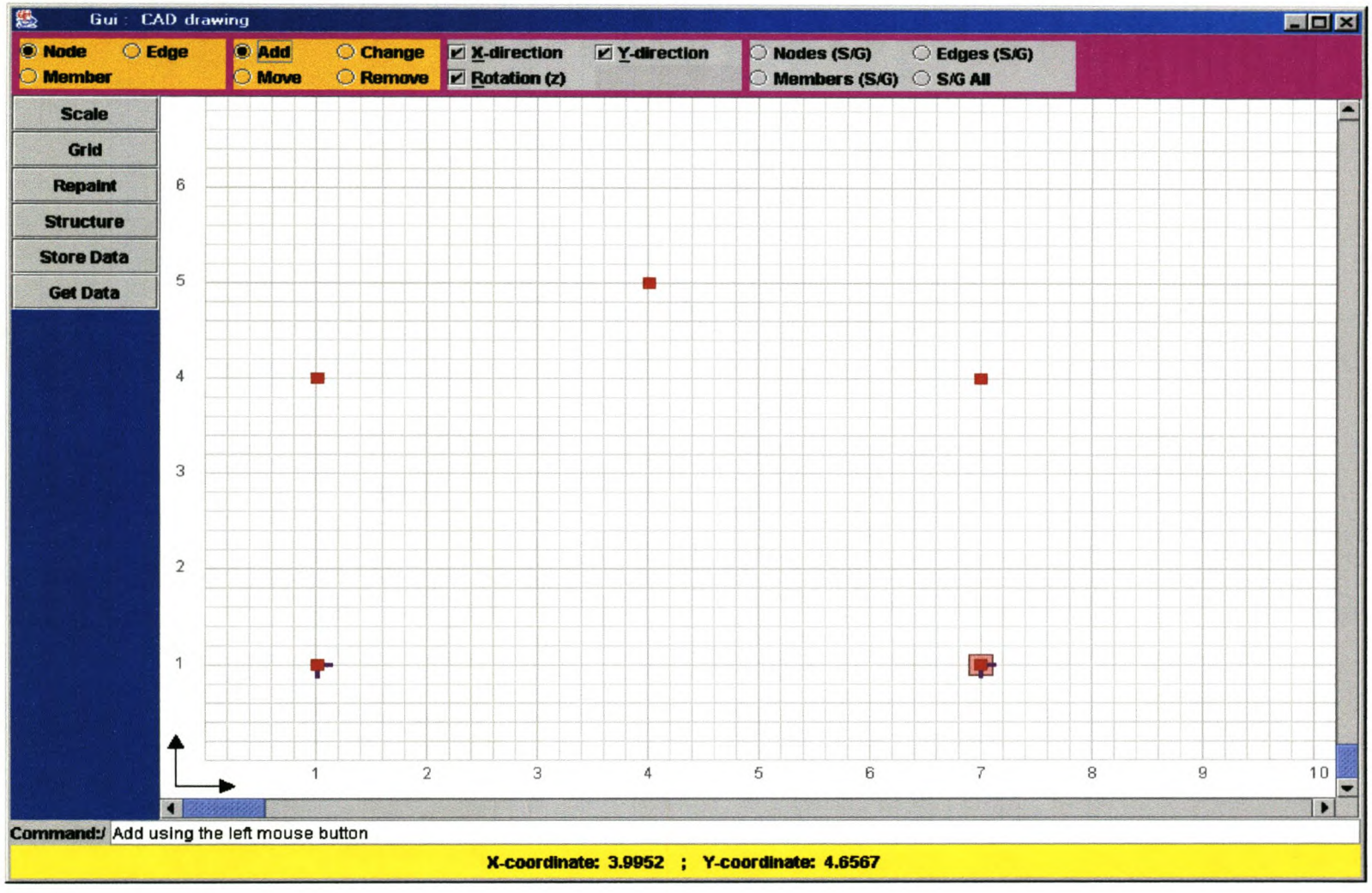


Figure 4.15

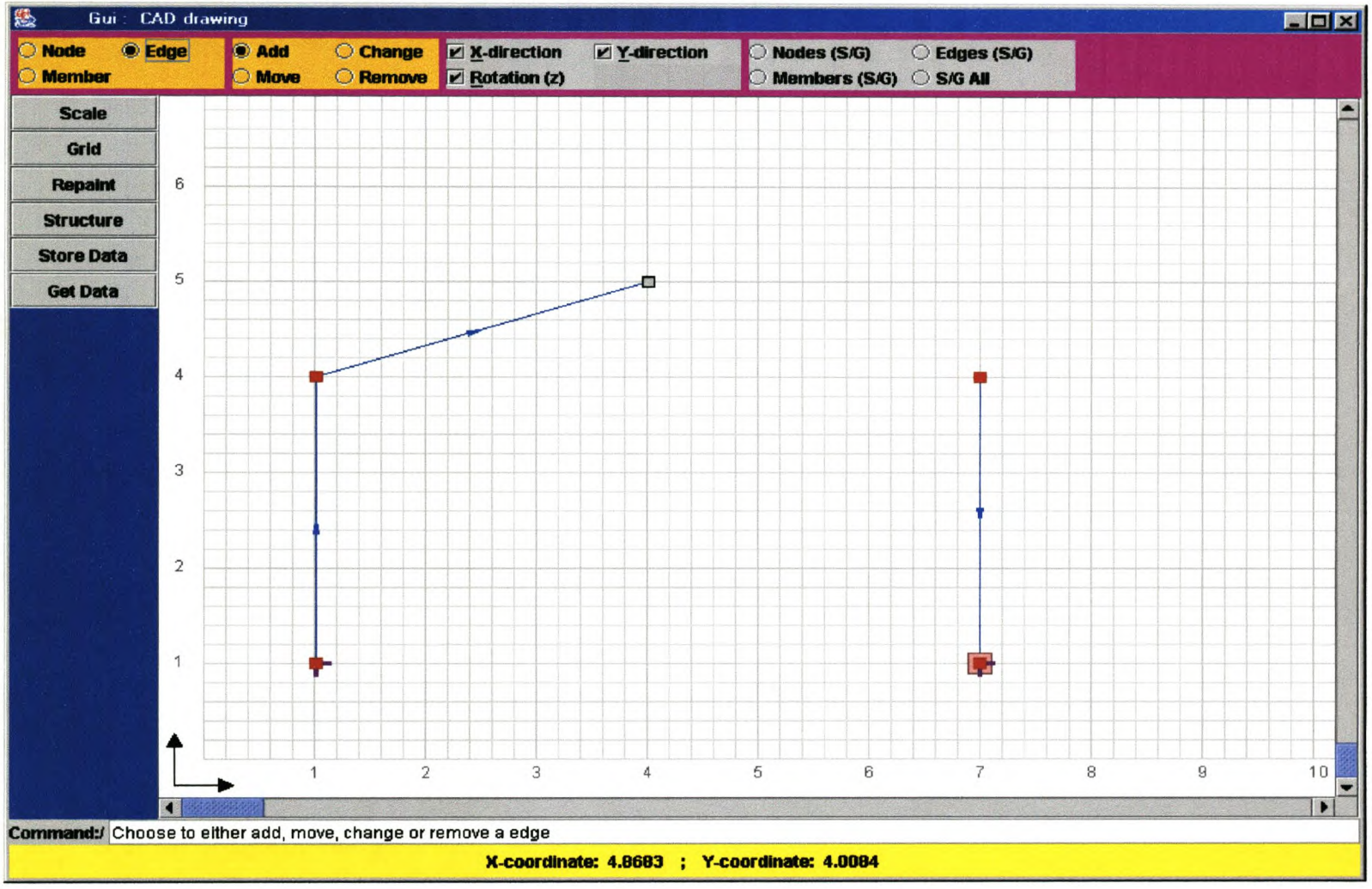


Figure 4.16

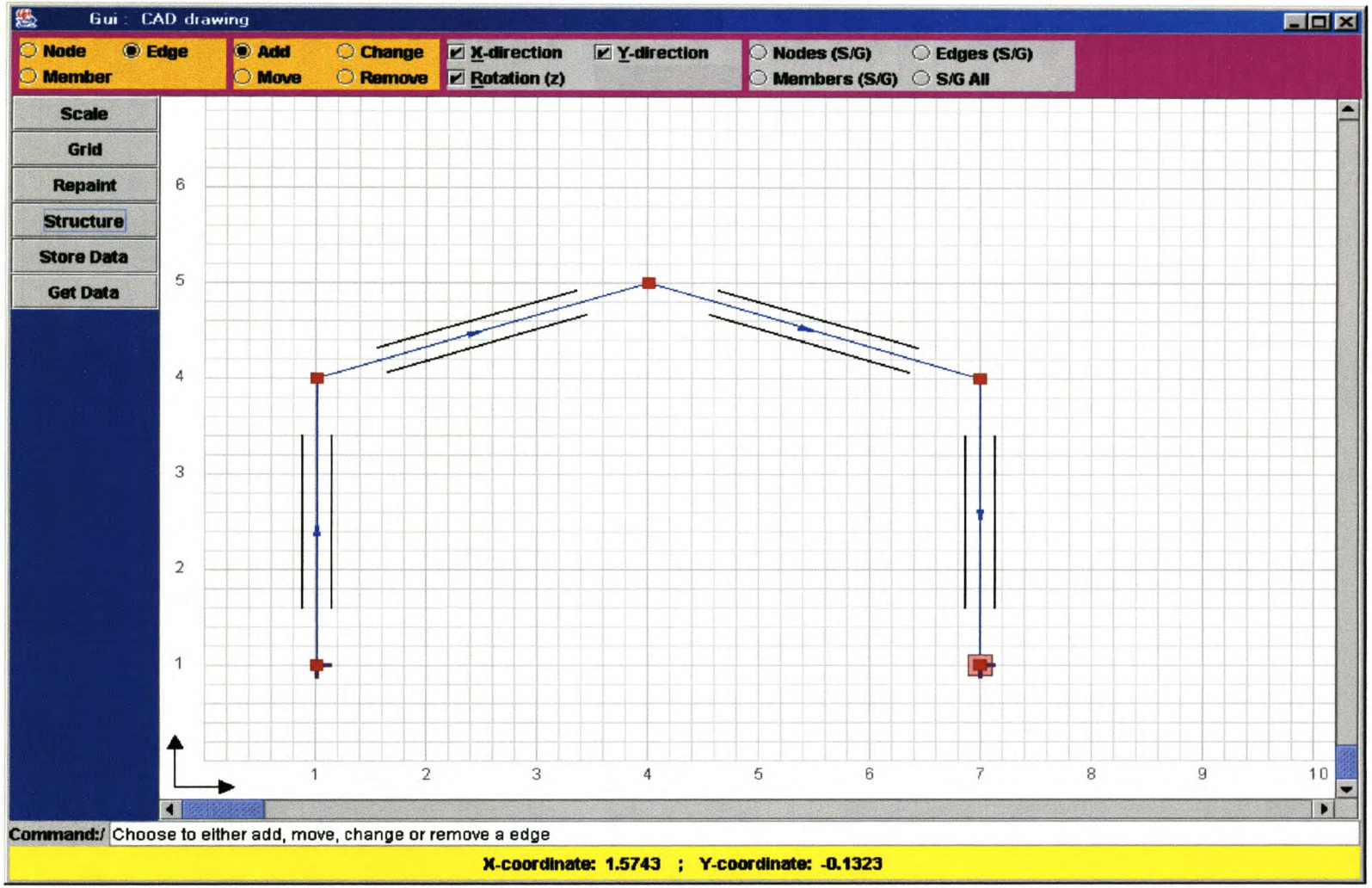


Figure 4.17

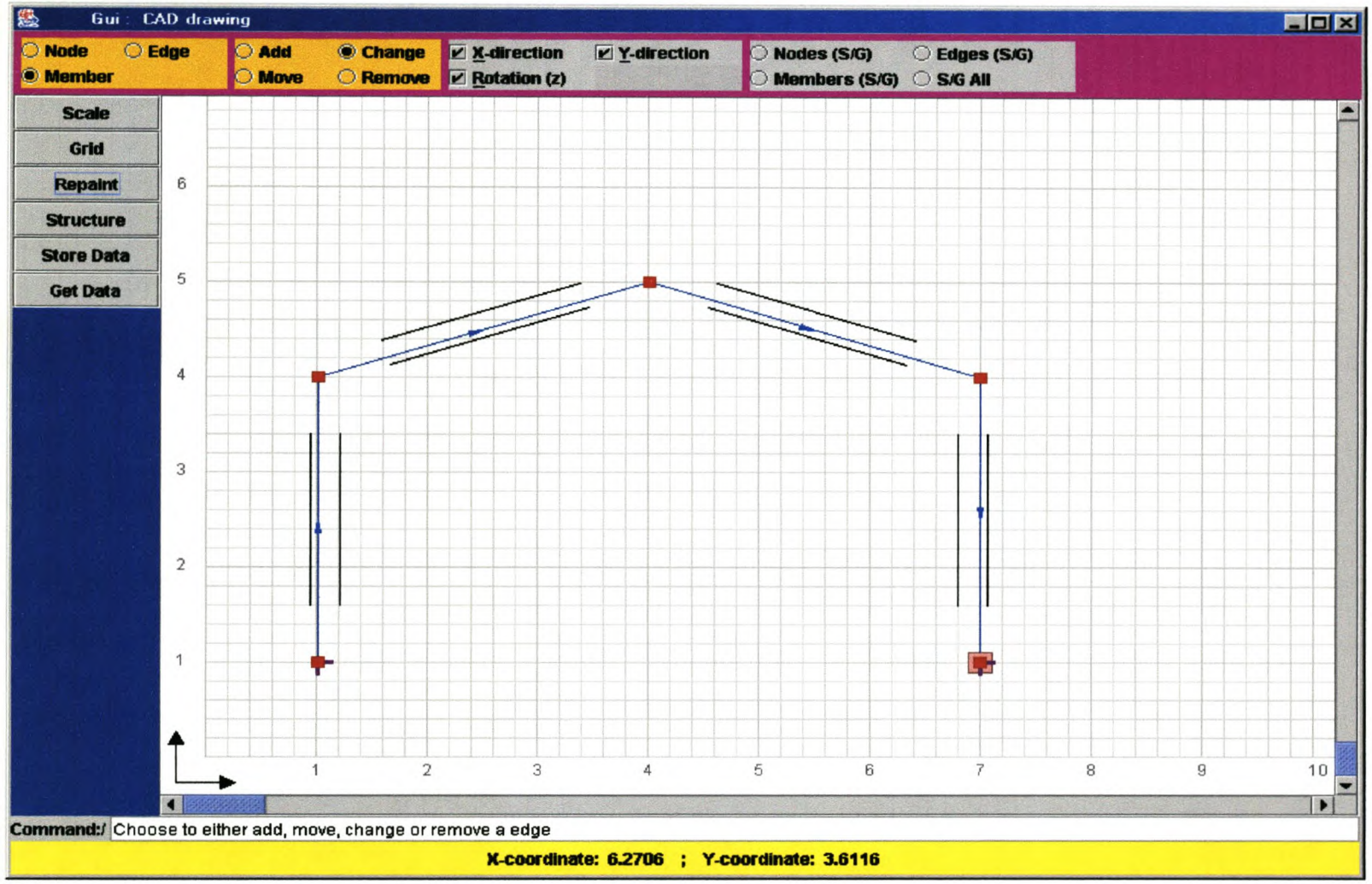


Figure 4.18

4.5 Future Developments

Future developments of the Graphical User Interface would include:

- The member class should be expanded so that it includes cross-sections of all the members needed to fully describe an industrial steel structure. The user should be given the option of choosing a member to be added to the reference frame. The additional members that need to be introduced are: rectangular sections, circular sections, T-type sections and angle sections.

- Members should also be described in more detail. Introducing holes, Figure 4.19, in members to describe connections with bolts would prove helpful. This will also become necessary when the connections between members are described in more detail.

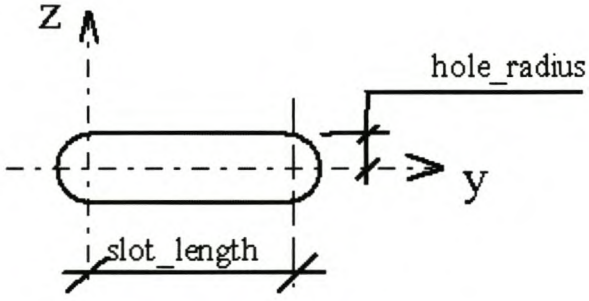


Figure 4.19: Representation of a hole.

- It is also important to introduce a class MemberEnd. This class would describe the geometry of a member end. This would make it possible to describe chamfers, notches and skewed ends as shown in Figure 4.20 to Figure 4.22.

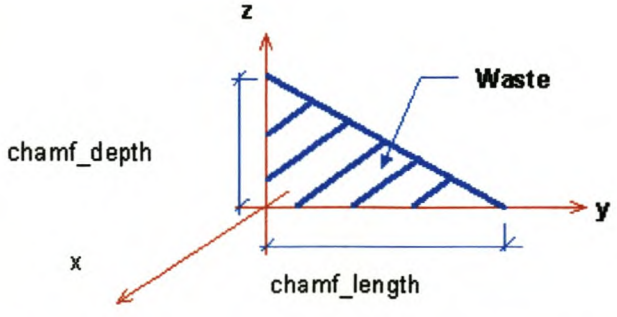


Figure 4.20: Representation of a chamfer.

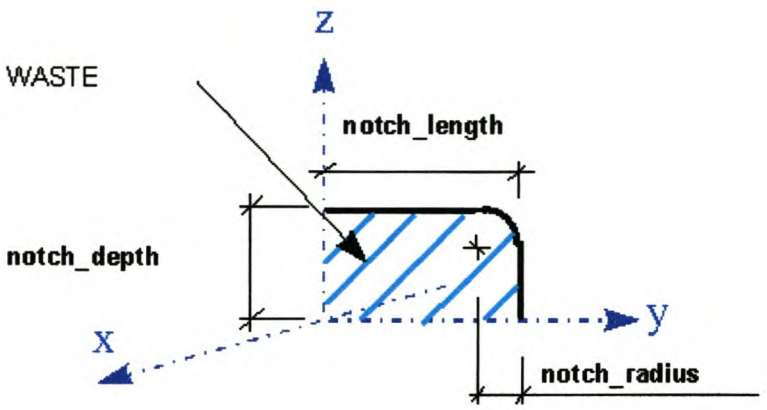


Figure 4.21: Representation of a notch.

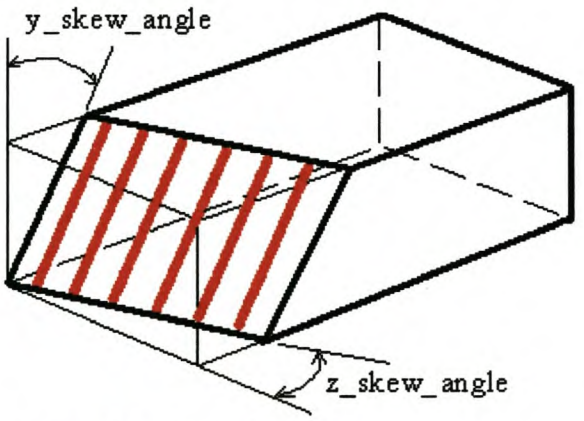


Figure 4.22: Representation of a skewed end.

- Together with the description of the member ends it is also necessary to describe the connections. This will be done by class Connections. This class will describe a plate by means of a polygon. The plate can then be used to describe connections as well as structural features such as a base plate.

- The description of pseudo-prismatic parts will also become necessary. Pseudo-prismatic parts are parts which vary linearly in cross section along the length of the element.

- Storing the entered data would be the main development for this graphical user interface. This would make it possible to create a drawing and to store it. This has the clear advantage of being able to continue with a project, and making appropriate changes without having to enter all the data from the start.

CHAPTER 5

AN INTERFACE TO THE PROJECT STEEL STRUCTURES DATABASE

5.1 Overview

Three basic components are used to describe the structural frame of a simple industrial steel structure in the graphical user interface as described in chapter 4. The three components are:

- nodes
- edges
- members.

These three components are therefore the entities that need to be stored within the database PSS. The STEP entities were subdivided to fully represent the different graphical user interface components. The mapping of the components to STEP entities is shown in Figure 5.2 and a key for the interpretation of the figure is shown in Figure 5.1. The entities were subdivided as follows:

- nodes were mapped to the STEP entities
  - Node
  - Point
- edges were mapped to the STEP entities
  - Element
- members were mapped to the STEP entities
  - GeometricRepresentation
  - PrismaticPartGeometry
  - SectionProfile
  - ITypeSect

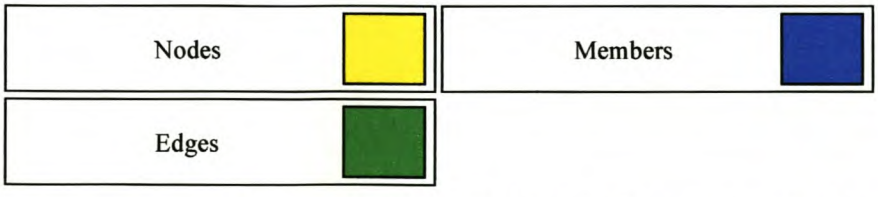


Figure 5.1: Key to Database implementation diagram

The exchange of data between the graphical user interface and the database is made possible with the use of database representatives.

The database representatives are instances of a set of classes called the database representative classes. These classes provide the interface to store and return data from the database. The data-representatives are instances that have the same selection identifiers and attributes as corresponding information units in the database. The data-representatives' primary tasks are to store themselves in the database and to retrieve their state from the database.

The database representative classes comprise three Java packages, all containing several classes. These packages are:

- DbJava
- AP230
- Linked

The package *DbJava* consist of Java classes that are responsible for opening a connection to the database and to store attributes from *CadDrawing* to the appropriate tables. These classes are also responsible for returning values stored within the database. The retrieval of values from the database and the instantiation of new *CadDrawing* objects is done with the help of the packages *AP230* and *Linked*.

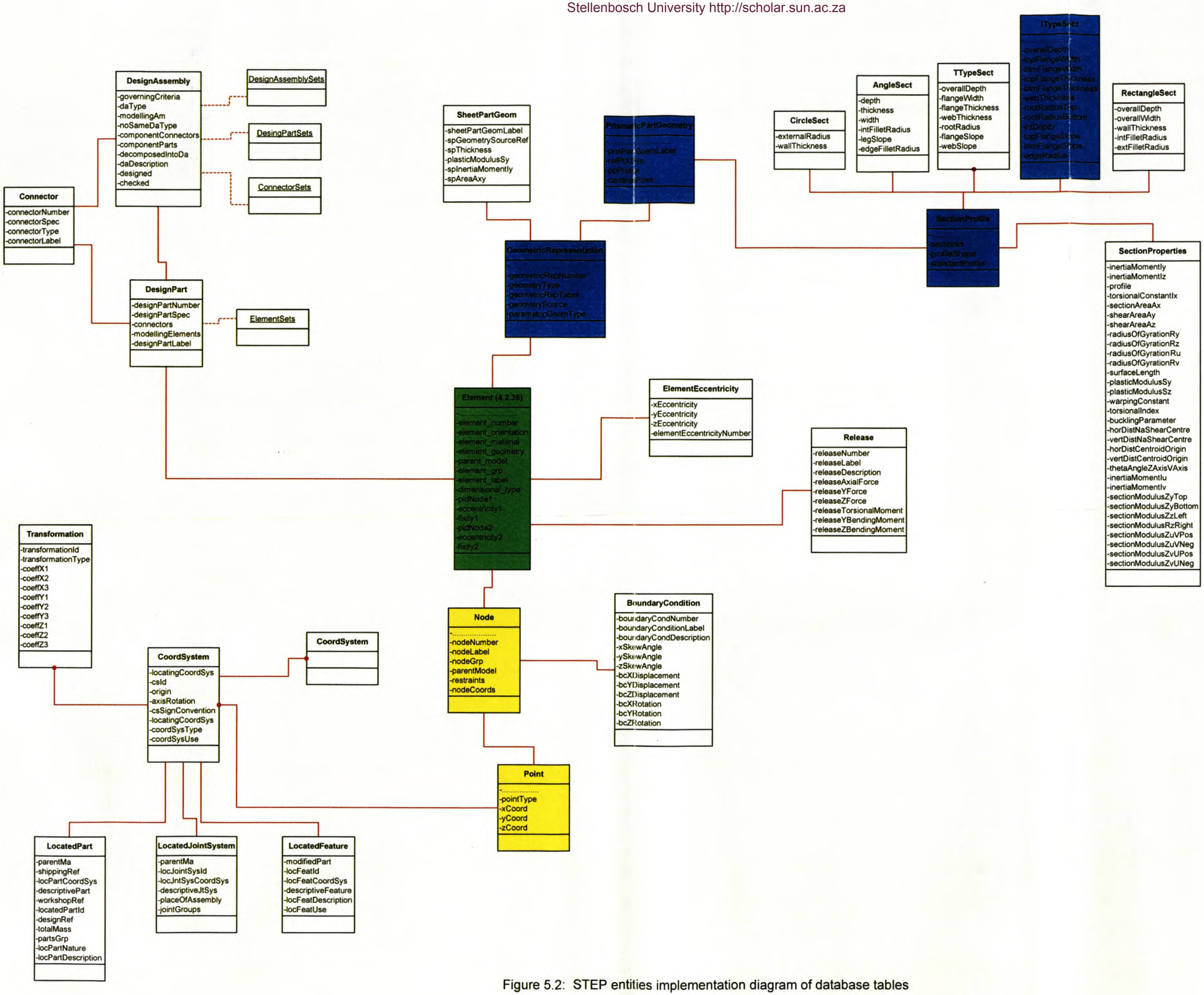


Figure 5.2: STEP entities implementation diagram of database tables

Figure 5.3 shows a diagram depicting how the different packages are used to store information to the database and also how information is retrieved once it has been stored.

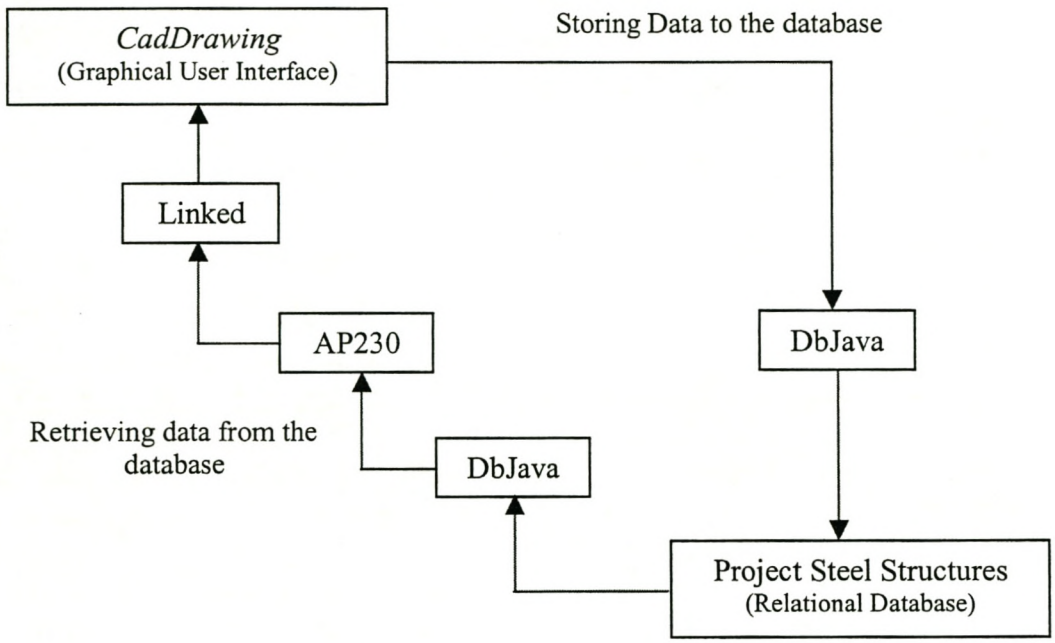


Figure 5.3: Different Software Components

5.2 Package DbJava

The Java package java.sql contains a number of classes which provide the middleware needed to handle a relational database. Basically, they allow one to establish a connection to a database and send queries. One can process the result of these queries, retrieve meta-information on the database, and handle the exceptions that occur.

The Java classes in package DbJava utilize the functionality provided in package java.sql. The main classes in this package are the following:

- Interface IpssComp
- Class PSScomm
- Class DbObject
- Class DatabasisException

As already mentioned there are three basic components (nodes, edges and members) used in the graphical user interface to describe a simple industrial steel structure. To simplify the storing of these components they were all made subclasses of the super-class *IdObject*. This allows one to add functionality and use methods from the super-class without redefining them in the subclasses.

The interface IpssComp must be implemented by classes which aim to store *IdObject* objects in the Project Steel Structure database (PSS) and retrieve *IdObject* attributes from PSS. IpssComp define the methods for storing and returning information to and from the database PSS.

The class PSScomm establishes a connection to the database PSS. It is also responsible for closing the connection, for executing updates as well as executing queries. This Java class is thus used for all communication with the database PSS.

The class DbObject contains a hash map of all the PSS component tables. These tables are the database tables which were created from the STEP entities. The PSS component table hash map contains all the references to the different tables. An example of this would be the table *tElement*. The reference to this table will be the keyword *Element*. One can now use the reference to exchange information with the appropriate table.

Class DbObject is also responsible for the entity handlers. For every table in PSS there is an entity handler to exchange data with its corresponding table. The different entity handlers are:

- Class DbElement
- Class DbGeometricRep
- Class DbITypeSect
- Class DbNode
- Class DbPoint
- Class DbPrismaticPartGeom
- Class DbSectionProfile.

These entity handlers are all Java classes and form part of the package DbJava. These entity handlers all implement the interface IpssComp. With this interface it is possible to execute *put* and *get* statements in order to either store or receive information from the database.

This means that the three components from the graphical user interface are identified, and the attributes associated with such a component handed on to the relevant entity handler. Storing the component node for example, two entity handlers will be called, DbNode for the table *tNode* and DbPoint for the table *tPoint*.

DbObject forms the highest class of the database representative's structure, since all other class applications concerning the representatives are executed from within this class. Executing class DbObject, from within CadDrawing, statements are used to connect with the appropriate Java classes depending on the graphical component that is to be stored or instantiated from the database.

An example of such an operation is the storing of a node in the database PSS. Once the class DbObject is executed a connection has to be established with the database in order to perform the necessary operations. This is done by executing the class PSScomm. Once the connection to the database has been opened the relevant IdObject can be stored. Class DbObject identifies for example that the IdObject is a Node object, and the appropriate entity handlers are called to handle the storing of the Node object. The two handlers called are, as mentioned above, DbNode and DbPoint.

Each entity handler receives the necessary information from the Node object and stores its information in the appropriate table. The entity handler DbNode stores its relevant attributes in table *tNode* and DbPoint stores its information in table *tPoint*. Once the information has been stored in the correct tables the

connection is closed and further operations can be carried out. All this is done from within the class DbObject.

5.3 Package AP230

The Java classes in this package are derived form the STEP protocol. Since all the tables in the database will have seven common attributes a super-class was created to define these attributes. The super-class in this package is ObjectAP230. It contains a persistent identifier, three attributes to define the versioning of the objects and three attributes to define the user as described in section 3.2 (Structure of a Standard entity table).

When data is restored form the database it is done by returning an object. Defining this object as an ObjectAP230 is advantageous since it represents all the subclasses as well. It also ensures that there will not be any problems joining classes as described in section 5.4 (Package Linked). Therefore it is not necessary to define each subclass when restoring data. It is possible to directly use methods defined in the subclasses. The subclasses of Object AP230 are:

- ElementAP230
- GeometricRepAP230
- ITypeSectAP230
- NodeAP230
- PointAP230
- PrismaticPartGeomAP230
- SectionProfileAP230

These subclasses contain the same attributes as defined in the STEP protocol. They also contain methods for returning information from the database and storing them in predefined variables. With these variables it is possible, e.g., to recreate graphical user interface (GUI) components.

5.4 Package Linked

Data from the database has to be used to recreate GUI components of the implementation CadDrawing. The information stored in the variables of instances of the different classes within package AP230 is assembled in this package to recreate the GUI components, namely Nodes, Edges and Members.

The classes in this package are the following:

- ElementEdge
- GeomPrisSectIType
- NodePoint

The information is assembled as follows:

- ElementEdge extracts data from:
  - ElementAP230
- GeomPrisSectIType extracts data from:
  - ElementAP230
  - GeometricRepAP230
  - PrismaticPartGeomAP230
  - SectionProfileAP230
  - ITypeSectAP230
- NodePoint extracts data from:
  - NodeAP230
  - PointAP230

5.5 Storing data in the Database PSS

Data is stored in a database table using any one of the appropriate entity handlers associated with that specific table. These handlers have put and get statements with which the database can be updated and queried. The put statement within the handlers is more or less the same for all handlers, with only the attributes differing. The same is also true for the get statements.

An IdObject can be a Node, Edge or a Member. When an IdObject needs to be stored the class DbObject selects the correct entity handler. The handler tries to cast the IdObject as a Node, Edge or Member depending on what needs to be stored. If the IdObject could be cast as the appropriate GUI component a put statement is executed from within the handlers. This happens automatically if the correct handler has been appointed, otherwise an error message will appear on screen.

The persistent identifier as well as the versioning and the access control of the IdObject is received from the object. The attributes of the chosen IdObject are received from the relevant object. This is done with the help of get-methods located within the chosen IdObject. In the handler DbElement an example of returning the attributes of the IdObject cast as an Edge, with the use of get methods found in class Edge, are as follows:

```
int elementNumber          = edge.getEdgeNumber();
double elementOrientation  = edge.getOrientation();
String dimensionalType     = edge.getDimension();
String pidNode1            = edge.getNodeId1();
String pidNode2            = edge.getNodeId2();
```

The methods perform the following actions:

- getEdgeNumber() returns the value for the element number of the specific element
- getOrientation() returns the value for the element orientation between the z-axis and the yz-plane
- getDimension() returns the dimension of the element as a 1D, 2D or 3D object
- getNodeId1() returns the persistent identifier for the start node connected

to the element

- getNodeId2() returns the persistent identifier for the end node connected to the element

These are the attributes that are stored in PSS. It is necessary to construct an SQL-statement in order to store the attributes to the database. It is of great importance that the sequence of parameters are the same as in the database table. Once the SQL-statement has been compiled the update query is executed and the attributes are stored in the appropriate column within the database table.

```
public String putElementSQL(String pid,……,String fixity2) {
    String lead = "INSERT INTO " + elementTable + " VALUES (";
    String c    = ",";
    String q    = " ' ";
    String putSQL =
    lead +
    q + pid + q                         + c +
    versionNumber                       + c +
    versionTime                         + c +
    q + versionHistory + q              + c +
    q + ownerId + q                     + c +
    q + taskgroupId + q                 + c +
    accessCode                          + c +
    elementNumber                       + c +
    elementOrientation                  + c +
    q + elementMaterial + q             + c +
    q + elementGeometry + q             + c +
    q + parentModel + q                 + c +
    q + elementLabel + q                + c +
    q + dimensionalType + q             + c +
    q + pidNode1 + q                    + c +
    eccentricity1                       + c +
    q + fixity1 + q                     + c +
    q + pidNode2 + q                    + c +
    eccentricity2                       + c +
```

```
        q + fixity2 + q                    +
        ")";
        return putSQL;
}//eo putElementSQL
```

Figures 5.3 to 5.5 show graphically how the different graphical user interface components are stored in the database PSS. It demonstrates which entity handlers, from the package DbJava, are needed to store each of the components and to which tables these handlers store the information in PSS.

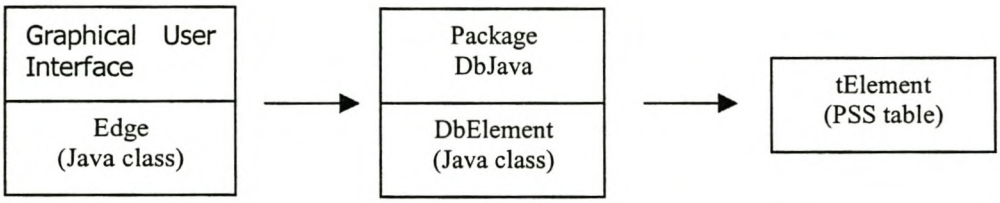


Figure 5.4: Storing GUI component Edge

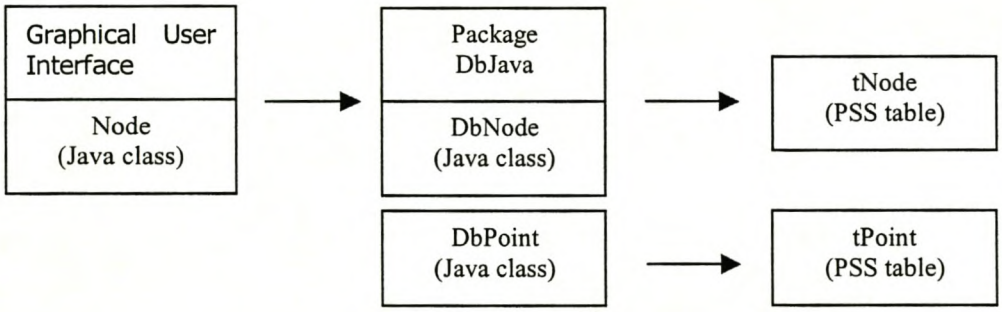


Figure 5.5: Storing GUI component Node

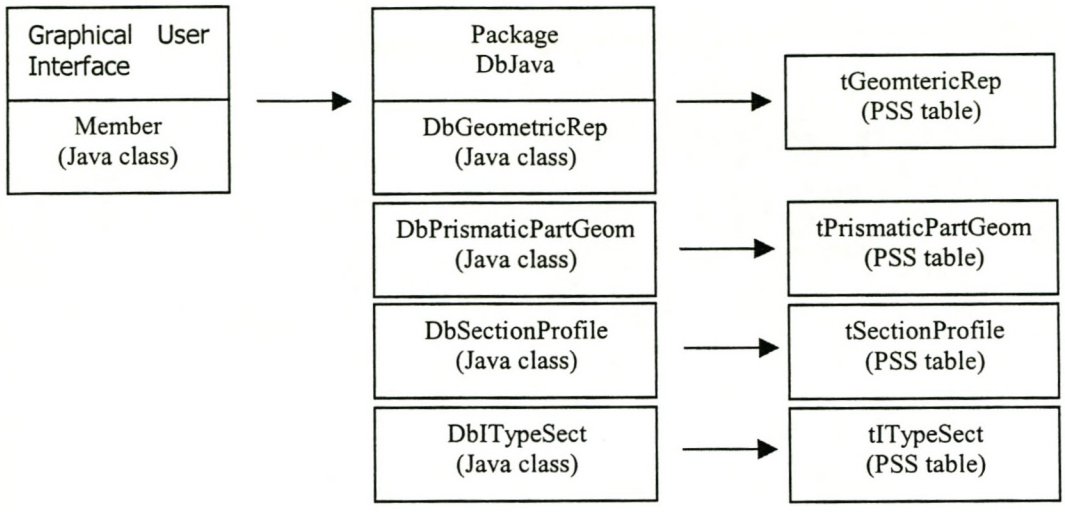


Figure 5.6: Storing GUI component Member

5.6 Reinstating IdObjects using data from the Database PSS

Information is retrieved from the database in the form of result sets, generated by executing get-statements of the classes of package DbJava, e.g.

public ObjectAP230 get(String pid,....., boolean cfMode) throws DatabasisException

The method name is the same for all the classes in package DbJava, due to the fact that ObjectAP230 is the super class of all the other classes in package AP230. As a result it is possible to condense and join the classes when new components are created.

This operation is controlled from the graphical user interface. Information is received from the database through SQL statements. These statements return result sets that are used to extract data form the appropriate fields. An example of setting up an SQL statement for returning a result set containing data for an element is shown below.

```
public String getElementSQL(String pid, int versionNumber) {
   String checkPID =
        "SELECT * " +
        "FROM (" + elementTable+ ") " +
        "WHERE ([pid] = '" + pid + "') " +
        "AND ([versionNumber] = " + versionNumber + ") ";
   return checkPID;
}//eo getElementSQL
```

The method generates an SQL statement which, when submitted as a query, returns a record set from the stated database table. The variables are the *pid* (persistent identifier), *versionNumber* (version number) and *elementTable* (PSS table name). The pid and versionNumber are specified by the user when the method is called. The elementTable variable is automatically specified within the class when the method is executed.

Once the SQL string has been constructed, the next step is to use this string to create the required result set:

```
ResultSet rs;
    String check = getElementSQL(pid, versionNumber);
    rs           = dbObject.pssComm.executeQuery(check);
```

Once the result set has been constructed, it is necessary to extract the data from the relevant fields. This is done by firstly instantiating an empty instance of the appropriate class in package AP230 and then populating it with the required variables. Methods of the class are used to store the data from the result set in variables contained in the class:

```
        ElementAP230 element = new ElementAP230();

// sets the value for the element number
        element.setElementNumber(rs.getInt("elementNumber"));
// sets the value for the element orientation between the z-axis and the yz-plane
        element.setOrientation(rs.getDouble("elementOrientation"));
// sets the type of material of the element
        element.setElementMaterial(rs.getString("elementMaterial"));
// sets the type of geometrical representation for the element
        element.setGeometry(rs.getString("elementGeometry"));
// sets the value for the analysis model of which the element is part
        element.setParent(rs.getString("parentModel"));
// sets the description of the element
        element.setLabel(rs.getString("elementLabel"));
// sets the value for the dimension of the element
        element.setDimension(rs.getString("dimensionalType"));
// sets the persistent identifier of the start node
        element.setPidNode1(rs.getString("pidNode1"));

// sets the value for the eccentricity of the start node
        element.setEcc1(rs.getDouble("eccentricity1"));
// sets the value for the type of release associated with the start node
        element.setFixity1(rs.getString("fixity1"));
// sets the persistent identifier of the end node
        element.setPidNode2(rs.getString("pidNode2"));
```

```
// sets the value of the eccentricity of the end node
        element.setEcc2(rs.getDouble("eccentricity2"));
// sets the value of the type of release associate with the end node
        element.setFixity2(rs.getString("fixity2"));
```

Once this operation has been successfully completed the data is in a form similar to the STEP protocol. This information is subsequently condensed to only include the variables needed to restore a GUI component. This is done using the classes in package Linked. A new object of the appropriate class of package Linked is created. The predefined variables of the object are allocated values as described in section 5.4:

```
ElementEdge eEdge          = new ElementEdge(element);

int     countOfEdge        = eEdge.countOfEdge;
String  id                 = eEdge.id;
String  node1Id            = eEdge.node1Id;
Stting  node2Id            = eEdge.node2Id;
int     edgeNumber         = eEdge.edgeNumber;
int     designPartNumber   = eEdge.designPartNumber;

Edge edge                  = new Edge(graphPanel, id, node1Id, node2Id);
edge.setEdgeNumber(edgeNumber);
edge.setDesignPartNumber(designPartNumber);
elementSet.add(edge);
```

Each recreated component is once again stored in a component set and one is able to recreate a previous structure representation.

A graphical representation of the above-mentioned detail is shown in Figures 5.7 to 5.9. The flow of information is shown, starting from the PSS tables, where the data was stored, to the packages DbJava, AP230 and Linked before a graphical user interface component is recreated from the stored information.

Figure 5.6 shows how an Edge is recreated.

Figure 5.7 shows how a Node is recreated.

Figure 5.8 shows how a Member is recreated.

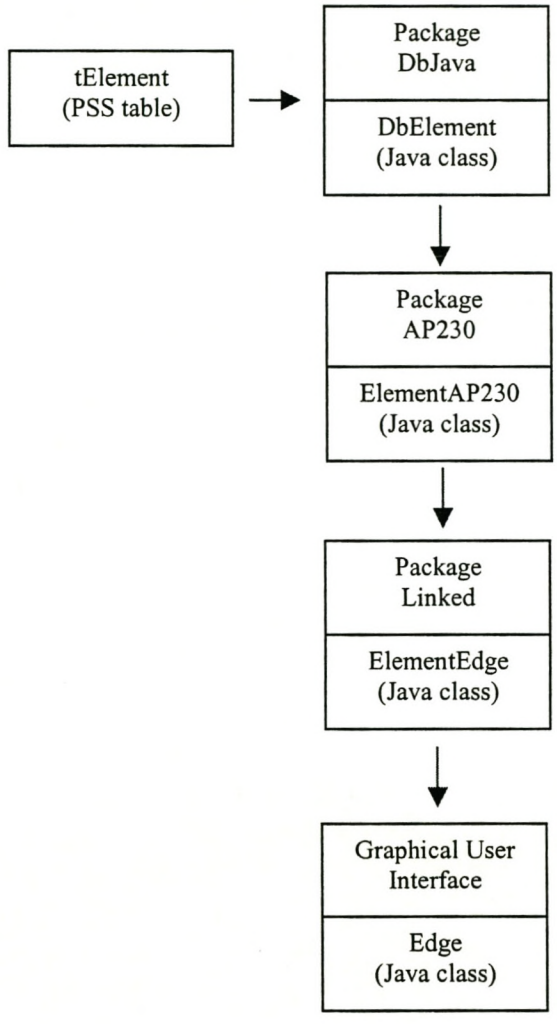


Figure 5.7: Restoring the GUI component Edge

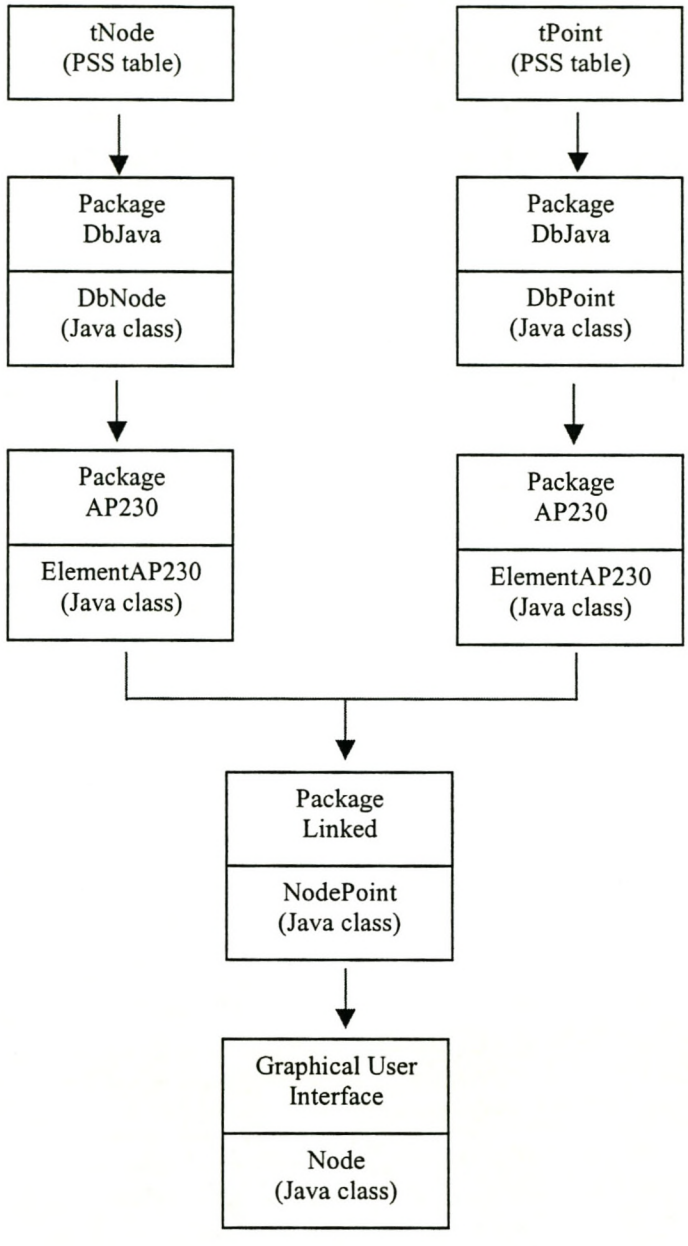


Figure 5.8: Restoring the GUI component Node

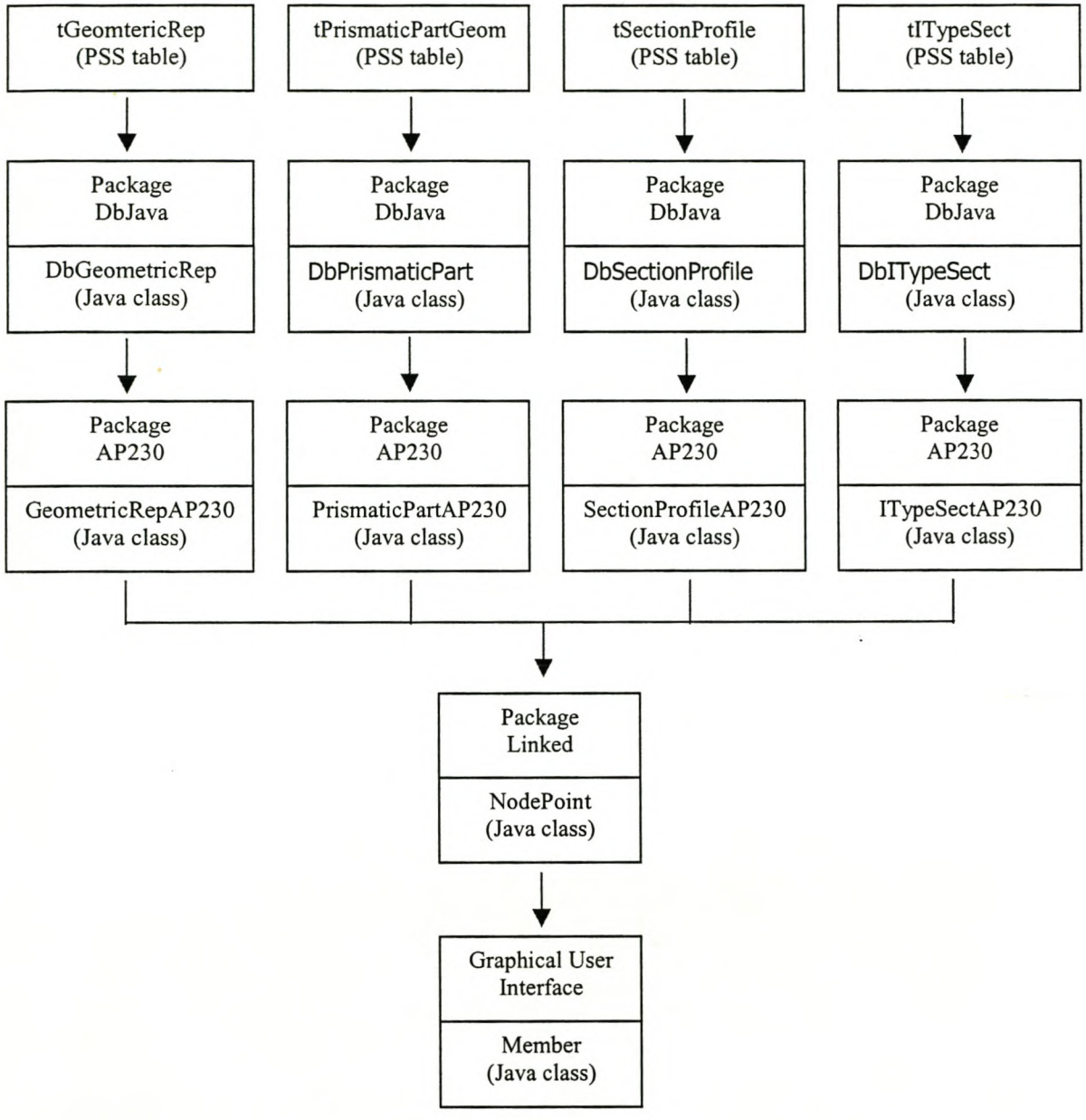


Figure 5.9: Restoring the GUI component Member

CHAPTER 6

CONCLUSIONS

The information contained in the STEP documents are very complex and difficult to interpret. It was possible, however, to selectively identify information needed for the completion of the thesis.

The following was achieved:

- A simplified subset of entities was identified which can be used to describe the geometry of a simple industrial steel structure, to analyse it, and to design it.
- A database model was created to which the data could be mapped.
- A simple application supporting the definition of a steel frame was created. This application is able to store and retrieve data to and from the database, in a generalized way. The application can be upgraded as discussed in section 4.5 (Future Developments).

Referring to the listed achievements, the conclusion can be drawn that the objectives of the thesis have been met. A foundation has also been laid for identification and inclusion of additional STEP entities, thereby extending the data model and the database. In this regard, some work has already been done, i.e. the STEP entities needed to describe the manufacturing of a simple industrial steel structure have been identified. Entities were also identified that are needed to describe the load combinations associated with a steel frame. Both sets of entities are shown graphically in Appendix B. Figure B.1 represents the manufacturing details needed to define a structure, and Figure B.2 represents the loads associated with a design assembly.

BIBLIOGRAPHY

1 *A Gentle Introduction to Relational and Object Oriented Database, Cambridge, UK, 1998,* www.uk.research.att.com/~fms/db

2 *Abell, Steven T., Using Java with JDBC and SQL,* *www.developer.netscape.com/docs/technote/database/sql*

3. Alley, Michael, *The Craft of Scientific Writing*, 3rd Edition, Springer, 1996

4 Anderson, Jonathan, Poole, Millicent., *Thesis and Assignment Writhing*, 2nd Edition, John Wiley & Sons, 1994

5 Booch, Grady, Rumbaugh, James, Jacobson, Ivar, *The Unified Modelling Language User Guide*, Addison Wesley Longman Inc., 1999

6 DEVx, *Object Oriented Databases are worth a Closer Look*, www.devx.com/dbzone/articles/sf0601/sf0601-1.asp

7 DEVx, *Using Object-Oriented Databases: A Step-B Tutorial*, www.devx.com/dbzone/articles/sf0701/sf0701-1.asp

8 Garrett, James H. Jr., Fenves, Steven J., Kiliccote, Han, *Evaluation of the AP230 Application Protocol*, NIST, 1997

9 Greenspan, Jay, *Your First Database*, www.hotwired.lycos.com/webmonkey/templates/print_template.html

10 Greenspun, Philip, *Database Management Systems*, www.arsdigita.com/books/panda/database-choosing

11 Hand, Steve, Chandler, Jane, *Introduction to Object-oriented Database*, 1998,

www.dis.port.ac.uk/~chandler/OOLectures/database/database.htm

12 Hardwick, Martin, *STEP Database Tutorial: Chapter 1 "Making Business Objects using EXPRESS-X"*, STEP Tools Inc., 1999

13 Hardwick, Martin, *STEP Database Tutorial: Chapter 2 "STEP Database Control Language"*, STEP Tools Inc., 1999

14 Hardwick, Martin, *EXPRESS-X Examples: EXPRESS-X Definition for AP203*, STEP Tools Incorporated, 1999, www.steptools.com/library/express-x/p203.html

15 Hardwick, Martin, *EXPRESS-X Examples: EXPRESS-X Legacy Database to AP203 CC1 Mapping*, STEP Tools Incorporated, 1999, www.steptools.com/library/express-x/hardwick1.html

16 Loffredo, David, *Efficient Database Implementation of EXPRESS Information Models*, PhD Thesis, Rensselaer Polytechnic Institute, Troy, New York, May 1998

17 McClure, Steve, *Object Database vs. Object-Relational Databases*, IDC Bulletin #14821E, 1997

18 OMG, *Cobra Basics*, Object Management Group Inc, 2001, www.omg.org/gettingstarted/cobrafaq.htm

19 OMG, *Introduction of OMG's Specification*, Object Management Group Inc, 2001 www.omg.org/gettingstarted/specintro.htm

20 OMG, *OMG Specification and Process: The Big Picture*, Object Management Group Inc, 2001, www.omg.org/gettingstarted/overview.htm

21 OMG, *What is COBRA?*, Object Management Group Inc, 2001, www.cgi.omg.org/cobra/whatiscobra.html

22 OMG, *Welcome to OMG's COBRA for Beginners Page*, Object Management Group Inc, 2000,

www.cgi.omg.org/cobra/beginners.html

23 PDES, *Recommended Practices for AP203*, PDES Inc., 1998

24 *Product Data Tools*,
www.ramp.scra.org/ap224_whatstep.html

25 *Reference: HTML CheatSheet*, Lycos Worldwide, 2002
www.hotwired.lycos.com/webmonkey/reference/html_cheatsheet,

26 *Relational Databases get Java Connectivity with CocoBase*, THOUGHT Inc, 2000/2001,
www.thoughtinc.com/cber adv.html

27 Riordan, Rebecca M., *Designing Relational Database Systems*, Microsoft Press, 1999

28 SCRA, *STEP Application Handbook*, SCRA, 2000

29 Sides, Charles H., *How to Write and Present Technical Information*, 2nd Edition, Cambridge University Press, 1991

30 Spruit, Sandor, *Reflections on Java, Beans, and Relational Databases*,
www.javaworld.com/javaworld/jw-09-1997/jw-09-reflections_p.html

31 *The HTML Basics*, Wired Digital Inc, 1994 – 1999,
www.hotwired.lycos.com/webmonkey/teachingtool/html.html

32 Underwood Jason, Alshawi, Mustafa, *Forecating building element maintenance within a integrated construction environment*, Department of Surveying, Salford University, Manchester, UK, Elsevier Science B.V., 2000

33 Watson, A.S., *CIS Product Developments*,
www.leeds.ac.uk/civil/research/cae/cis/cis.htm

34 Watson, A.S., *Computer Aided Engineering Group*,

www.leeds.ac.uk/civil/research/cae/cae.htm

35 Watson, A.S., Leonard, D., *Eureka Project 130 CIMsteel*, www.leeds.ac.uk/civil/research/cae/cimsteel/cimsteel.htm, Leeds University

36 Yao, Yin-Ho, Trappey, Amy J.C., *ISO10303 compatible data model and its applications for PC configuration management*, Department of Industrial Engineering & Engineering Management, National Tsing Hua University, Hsinchu, Taiwan, ROC, Elsevier Science Ltd, 2000

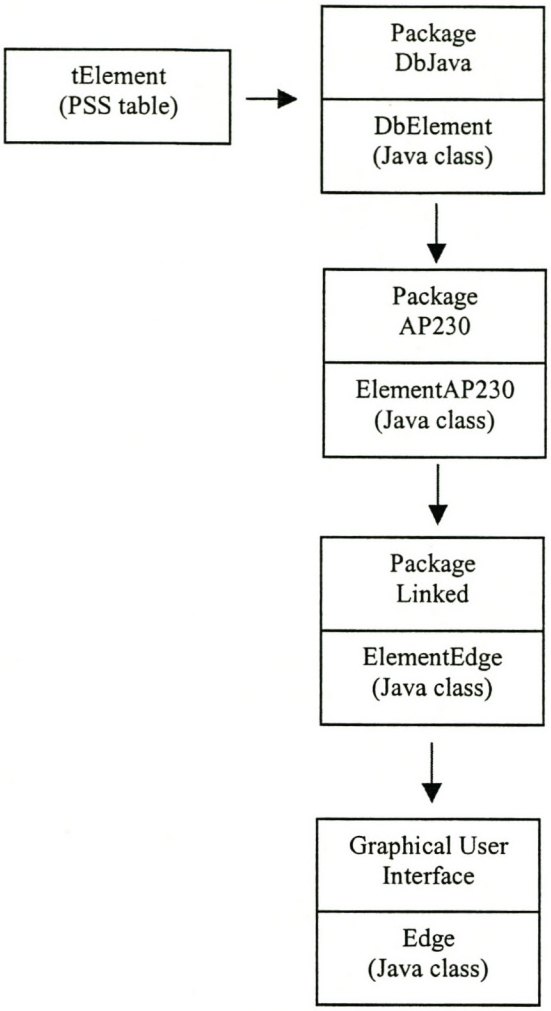


Figure 5.7: Restoring the GUI component Edge

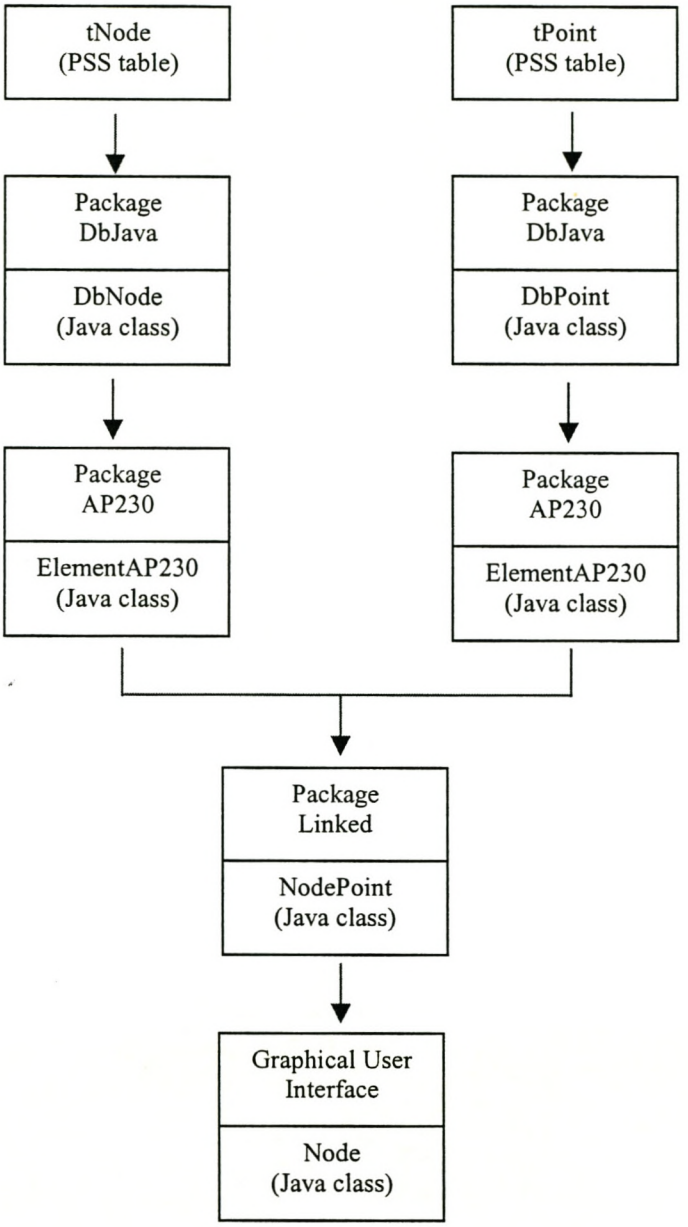


Figure 5.8: Restoring the GUI component Node

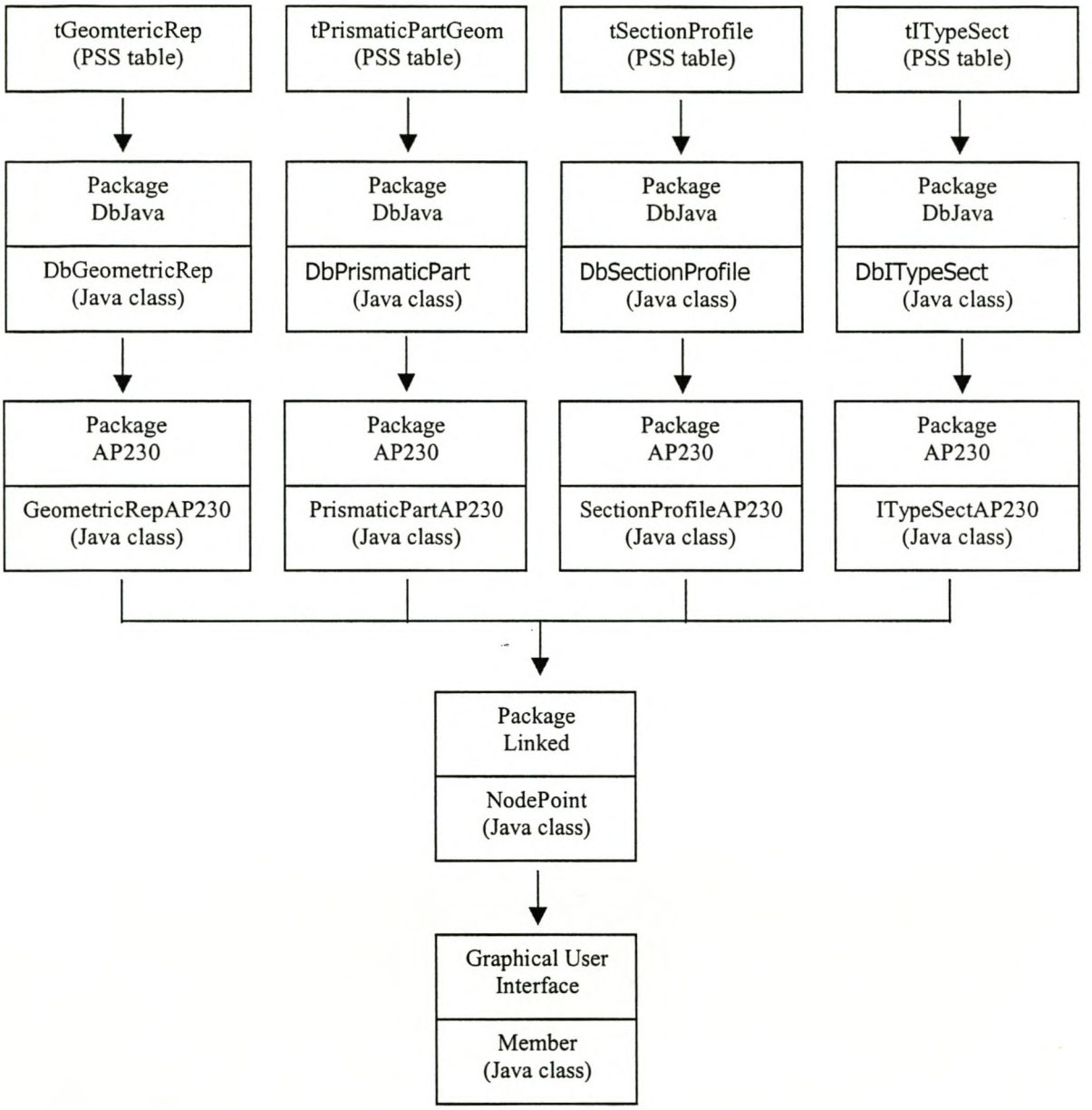


Figure 5.9: Restoring the GUI component Member

CHAPTER 6

CONCLUSIONS

The information contained in the STEP documents are very complex and difficult to interpret. It was possible, however, to selectively identify information needed for the completion of the thesis.

The following was achieved:

- A simplified subset of entities was identified which can be used to describe the geometry of a simple industrial steel structure, to analyse it, and to design it.
- A database model was created to which the data could be mapped.
- A simple application supporting the definition of a steel frame was created. This application is able to store and retrieve data to and from the database, in a generalized way. The application can be upgraded as discussed in section 4.5 (Future Developments).

Referring to the listed achievements, the conclusion can be drawn that the objectives of the thesis have been met. A foundation has also been laid for identification and inclusion of additional STEP entities, thereby extending the data model and the database. In this regard, some work has already been done, i.e. the STEP entities needed to describe the manufacturing of a simple industrial steel structure have been identified. Entities were also identified that are needed to describe the load combinations associated with a steel frame. Both sets of entities are shown graphically in Appendix B. Figure B.1 represents the manufacturing details needed to define a structure, and Figure B.2 represents the loads associated with a design assembly.

BIBLIOGRAPHY

1 *A Gentle Introduction to Relational and Object Oriented Database, Cambridge, UK, 1998,* www.uk.research.att.com/~fms/db

2 *Abell, Steven T., Using Java with JDBC and SQL,* *www.developer.netscape.com/docs/technote/database/sql*

3. Alley, Michael, *The Craft of Scientific Writing*, 3rd Edition, Springer, 1996

4 Anderson, Jonathan, Poole, Millicent., *Thesis and Assignment Writhing*, 2nd Edition, John Wiley & Sons, 1994

5 Booch, Grady, Rumbaugh, James, Jacobson, Ivar, *The Unified Modelling Language User Guide*, Addison Wesley Longman Inc., 1999

6 DEVx, *Object Oriented Databases are worth a Closer Look*, www.devx.com/dbzone/articles/sf0601/sf0601-1.asp

7 DEVx, *Using Object-Oriented Databases: A Step-B Tutorial*, www.devx.com/dbzone/articles/sf0701/sf0701-1.asp

8 Garrett, James H. Jr., Fenves, Steven J., Kiliccote, Han, *Evaluation of the AP230 Application Protocol*, NIST, 1997

9 Greenspan, Jay, *Your First Database*, www.hotwired.lycos.com/webmonkey/templates/print_template.html

10 Greenspun, Philip, *Database Management Systems*, www.arsdigita.com/books/panda/database-choosing

11 Hand, Steve, Chandler, Jane, *Introduction to Object-oriented Database*, 1998,

www.dis.port.ac.uk/~chandler/OOLectures/database/database.htm

12 Hardwick, Martin, *STEP Database Tutorial: Chapter 1 "Making Business Objects using EXPRESS-X"*, STEP Tools Inc., 1999

13 Hardwick, Martin, *STEP Database Tutorial: Chapter 2 "STEP Database Control Language"*, STEP Tools Inc., 1999

14 Hardwick, Martin, *EXPRESS-X Examples: EXPRESS-X Definition for AP203*, STEP Tools Incorporated, 1999, www.steptools.com/library/express-x/p203.html

15 Hardwick, Martin, *EXPRESS-X Examples: EXPRESS-X Legacy Database to AP203 CC1 Mapping*, STEP Tools Incorporated, 1999, www.steptools.com/library/express-x/hardwick1.html

16 Loffredo, David, *Efficient Database Implementation of EXPRESS Information Models*, PhD Thesis, Rensselaer Polytechnic Institute, Troy, New York, May 1998

17 McClure, Steve, *Object Database vs. Object-Relational Databases*, IDC Bulletin #14821E, 1997

18 OMG, *Cobra Basics*, Object Management Group Inc, 2001, www.omg.org/gettingstarted/cobrafaq.htm

19 OMG, *Introduction of OMG's Specification*, Object Management Group Inc, 2001 www.omg.org/gettingstarted/specintro.htm

20 OMG, *OMG Specification and Process: The Big Picture*, Object Management Group Inc, 2001, www.omg.org/gettingstarted/overview.htm

21 OMG, *What is COBRA?*, Object Management Group Inc, 2001, www.cgi.omg.org/cobra/whatiscobra.html

22 OMG, *Welcome to OMG's COBRA for Beginners Page*, Object Management Group Inc, 2000,

www.cgi.omg.org/cobra/beginners.html

23 PDES, *Recommended Practices for AP203*, PDES Inc., 1998

24 *Product Data Tools*,
www.ramp.scra.org/ap224_whatstep.html

25 *Reference: HTML CheatSheet*, Lycos Worldwide, 2002
www.hotwired.lycos.com/webmonkey/reference/html_cheatsheet,

26 *Relational Databases get Java Connectivity with CocoBase*, THOUGHT Inc, 2000/2001,
www.thoughtinc.com/cber adv.html

27 Riordan, Rebecca M., *Designing Relational Database Systems*, Microsoft Press, 1999

28 SCRA, *STEP Application Handbook*, SCRA, 2000

29 Sides, Charles H., *How to Write and Present Technical Information*, 2nd Edition, Cambridge University Press, 1991

30 Spruit, Sandor, *Reflections on Java, Beans, and Relational Databases*,
www.javaworld.com/javaworld/jw-09-1997/jw-09-reflections_p.html

31 *The HTML Basics*, Wired Digital Inc, 1994 – 1999,
www.hotwired.lycos.com/webmonkey/teachingtool/html.html

32 Underwood Jason, Alshawi, Mustafa, *Forecating building element maintenance within a integrated construction environment*, Department of Surveying, Salford University, Manchester, UK, Elsevier Science B.V., 2000

33 Watson, A.S., *CIS Product Developments*,
www.leeds.ac.uk/civil/research/cae/cis/cis.htm

34 Watson, A.S., *Computer Aided Engineering Group*,

www.leeds.ac.uk/civil/research/cae/cae.htm

35 Watson, A.S., Leonard, D., *Eureka Project 130 CIMsteel*, www.leeds.ac.uk/civil/research/cae/cimsteel/cimsteel.htm, Leeds University

36 Yao, Yin-Ho, Trappey, Amy J.C., *ISO10303 compatible data model and its applications for PC configuration management*, Department of Industrial Engineering & Engineering Management, National Tsing Hua University, Hsinchu, Taiwan, ROC, Elsevier Science Ltd, 2000

APPENDIX A

IDENTIFIED ENTITIES AND ASSOCIATED ATTRIBUTES NECESSARY TO DEFINE A SIMPLE INDUSTRIAL STEEL STRUCTURE:

1. DesignAssembly:

A DesignAssembly is a type of *Assembly* that defines a subset of a *Structure*. A DesignAssembly can be any node in the decomposition hierarchy generated by the design process. The design decomposition hierarchy need not correspond to the physical decomposition hierarchy of a structure, and because it allows different levels of descriptive detail, it provides for top-down design approaches. A DesignAssembly consists of a *StrucConnection*, *StrucMember* or a *StrucFrame*.

An Assembly provides a representation of a Structure for design or manufacturing purposes. An Assembly is either a DesignAssembly or a ManufacturingAssembly.

A Structure on the other hand provides the root node of two hierarchies - corresponding to decomposition for design purposes and to physical decomposition. A Structure is a discrete structural unit on one or more sites.

A StrucConnection, StrucMember and a StrucFrame are all a type of DesignAssembly that allows for the explicit categorization of a design assembly, as a structural connection, a structural member, or a structural frame for analysis or member design purposes.

Data associated with DesignAssembly:

- governingCriteria
  - specifies a set of DesignCriteria associated with a DesignAssembly
  - need not be specified for a particular DesignAssembly
  - application assertion
    - each DesignAssembly is governed by zero, one or many DesignCriteria
    - each DesignCriteria object governs zero, one or many DesignAssembly objects

- daType
  - specifies the type of DesignAssembly as StructuralFrame, or StructuralConnection, or StructuralMember, or undefined

- modellingAm
  - specifies an analysis model that is used to model a DesignAssembly
  - need not be specified for a particular DesignAssembly
  - application assertion
    - each DesignAssembly is modelled by one or many AnalysisModel objects
    - each AnalysisModel models zero, one or many DesignAssembly objects

- noSameDaType
  - specifies a value for the number of DesignAssembly objects that are of the same type
  - this attribute may be used for checking the consistency of data
  - need not be specified for a particular DesignAssembly

- componentConnectors
  - specifies a connector that is associated with a DesignAssembly
  - need not be specified for a particular DesignAssembly
  - application assertion
    - each DesignAssembly includes one or many Connector objects
    - each Connector is part of zero, one or many DesignAssembly objects

- componentParts
  - specifies a design part that is associated with a Design Assembly
  - need not be specified for a particular Design Assembly
  - application assertion
    - each Design Assembly is made up of one or many Design Part objects
    - each Design_part is part of zero, one or many Design_assembly objects

- decomposesIntoDa
  - specifies a(nother) DesignAssembly that forms part of a DesignAssembly
  - need not be specified for a particular DesignAssembly
  - application assertion
    - each DesignAssembly decomposes into one or many DesignAssembly objects
    - each DesignAssembly forms part of zero, one or many DesignAssembly objects

- daDescription
  - specifies a text description of a DesignAssembly
  - need not be specified for a particular DesignAssembly

- designed
  - specifies whether or not a DesignAssembly has been designed, and thus provides information relating to the status of a design

- checked

- specifies whether or not a DesignAssembly has been checked, and thus provides information relating to the status of a design

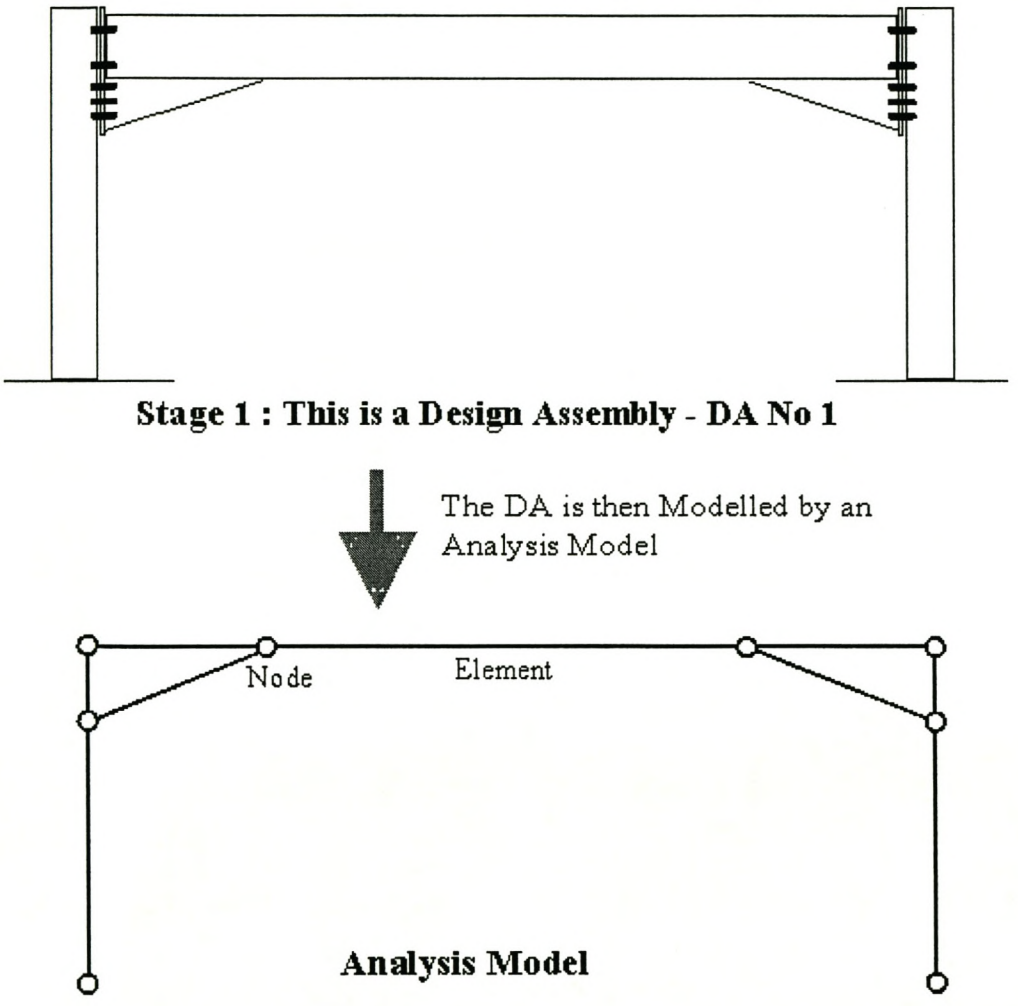


Figure A.1: Converting a Design Assembly into an Analysis Model

2. DesignPart:

A DesignPart is a representation of a physically *LocatedPart* (36) for member design purposes. Its representation may or may not correspond directly to an analytical *Element* (10).

Data associated with DesignPart:

- designPartNumber
  - specifies a unique numerical identifier for a DesignPart

- designPartSpec
  - specifies a specific part and thus the physical, material, and geometrical characteristics of a DesignPart
  - need not be specified for a particular DesignPart
  - application assertion
    - each DesignPart is described by one SPart
    - each SPart describes zero, one or many DesignPart objects

- connectors
  - specifies a connector associated with a DesignPart
  - need not be specified for a particular DesignPart
  - application assertion
    - each DesignPart is connected (to another DesignPart) by one or many Connector objects
    - each Connector connects zero, one or many DesignPart objects

- modellingElements
  - specifies a set of analytical elements used to model a DesignPart
  - need not be specified for a particular DesignPart
  - application assertion
    - each DesignPart is modelled by one or many Element objects
    - each Element models zero, one or many DesignPart objects

- designPartLabel
  - specifies a short text description of a part used in conjunction with a designPartNumber to uniquely identify a part
  - need not be specified for a particular DesignPart

3. SPart:

A structural part is a type of *SteelStructuralFrameEntity* and defines the geometry of *LocatedParts* (). It can be either a *PrismaticPart* (6), *PseudoPrismaticPart* (8), or a *SheetPart* (4).

Data associated with SPart:

- manufacturersRef
  - specifies a text description of a source where an explicit specification is to be found.
  - need not be specified for a particular SPart

- fabricationMethod
  - specifies the fabrication method for a specific part as:
    - rolled
    - cast
    - welded
    - cold-formed
  - need not be specified for a particular SPart

- sPartId
  - specifies a unique alpha-numeric identifier for SPart

- codeRef
  - specifies an alpha-numeric identifier for a code of practice or standard relating to a SPart
  - need not be specified for a particular SPart

- sPartMaterial
  - specifies the material used in the production of a specific part
  - need not be specified for a particular SPart
  - application assertion
    - each SPart has one Material
    - each Material is associated with zero, one or many SPart objects

- sPartType
  - specifies the type of specific part as
    - prismatic
    - pseudo-prismatic
    - sheet

4. SheetPart :

A sheet part is a type of structural part. It has a uniform geometry in cross-section. It's rectangular in shape, and can be defined by means of its width and length. The width and length may be defined as "absolute" (measured from the origin of the coordinate system associated with the part) or "relative" (measured from points, along the axes of the coordinate system associated with the part, which are offset by a given distance from the origin of the coordinate system). The origin of the coordinate system lies at the corner of the part. It is described in the yz-plane of the coordinate system while the profile is described in the xy-plane

Data associated with SheetPart:

- spPartRf
  - specifies a text description of a source where an explicit specification is to be found
  - need not be specified for a particular SheetPart

- spGeometry
  - specifies the geometry of a SheetPart
  - need not be specified for a particular SheetPart
  - application assertion
    - each SheetPart has its geometry specified by one SpeetPartGeom
    - each SheetPartGeom specifies the geometry for zero, one or many SheetPart objects

- yDimension
  - specifies a numerical value for the width of a sheet part measured in the y-axis
  - need not be specified for a particular SheetPart

- zDimention
  - specifies a numerical value for the length of a sheet part measured in the z- axis
  - need not be specified for a particular SheetPart

- massPerArea
  - specifies a numerical value for the mass per unit area of a SheetPart
  - need not be specified for a particular SheetPart

- relativeY
  - specifies whether or not a sheet part y-dimension is given relative to the origin of the coordinate system associated with the part
  - need not be specified for a particular SheetPart

- relativeZ
  - specifies whether or not a sheet part z-dimension is given relative to the origin of the coordinate system associated with the part
  - need not be specified for a particular SheetPart

- yOffset
  - specifies a numerical value for the offset applicable to a relative y-dimension
  - need not be specified for a particular SheetPart

- zOffset
  - specifies a numerical value for the offset applicable to a relative z-dimension
  - need not be specified for a particular SheetPart

5. SheetPartGeom:

A sheet part geometry is a type of GeometricRepresentation (9). It defines the cross-sectional geometry of a sheet part in the xy-plane of the coordinate system relating to a sheet part.

Data associated with a SheetPartGeom:

- sheetPartGeomLabel
  - specifies a short text description that may be used in conjunction with the geometricRepNumber of the GeometricRepresentation to uniquely identify a SheetPartGeometry
  - need not be specified for a particular SheetPartGeom

- spGeometrySourceRef
  - specifies a text description of a source where an explicit specification for a sheet part geometry is to be found.
  - need not be specified for a particular SheetPartGeom

- spThickness
  - specifies a numerical value for the thickness of a SheetPart
  - need not be specified for a particular SheetPartGeom

- plasticModulusSy
  - specifies a numerical value for the plastic modulus about the major (y) axis for a unit width (y-dimension) of a SheetPart
  - need not be specified for a particular SheetPartGeom

- spInertiaMomentIy
  - specifies a numerical value for the bending moment of inertia (second moment of inertia) about the major (y) axis for the width (y-dimention) of a SheetPart
  - need not be specified for a particular SheetPartGeom

- spAreaAxy
  - specifies a numerical value for the cross-sectional area, in the xy-plane, of a unit width (y-dimension) for a SheetPart
  - need not be specified for a particular SheetPartGeom

6. PrismaticPart:

Is a type of structural part. It has a uniform geometry along its length and can be defined as either "absolute" (measured from the origin of the coordinate system associated with a part) or "relative" (measured from some point, along the x-axis of the coordinate system associated with the part, which is offset by a given distance from the origin of the coordinate system). The origin of coordinate system relating to a prismatic part lies at the centroid of the face at one end of the part.

Data associated with PrismaticPart:

- ppSourceRef
  - specifies a text description of a source where an explicit specification for a PrismaticPart is to be found
  - need not be specified for a particular PrismaticPart

- ppLengthOffset
  - specifies a numerical value for the offset applicable to a relative length
  - need not be specified for a particular PrismaticPart

- ppLength
  - specifies a numerical value for the length of a PrismaticPart
  - need not be specified for a particular PrismaticPart

- ppLengthRelative
  - specifies whether or not the start of a given prismatic part length is placed relative to the origin of the coordinate system associated with the part.
  - need not be specified for a particular PrismaticPart

- sectionClassification
  - specifies a numerical identifier for the classification of the PrismaticPart
  - need not be specified for a particular PrismaticPart

- massPerLength
  - specifies a numerical value for the mass per unit length of a PrismaticPart.
  - need not be specified for a particular PrismaticPart

- ppGeometry
  - specifies a prismatic part geometry for a PrismaticPart
  - need not be specified for a particular PrismaticPart
  - application assertion

- each PrismaticPart is described by one PrismaticPartGeometry
- each PrismaticPartGeometry describes zero, one or many PrismaticPart objects

7. PrismaticPartGeometry:

A prismatic part geometry is a type of GeometricRepresentation (9). It defines the cross-sectional geometry of a PrismaticPart and a PseudoPrismaticPart in the yz-plane of the coordinate system relating to a part

Data associated with PrismaticPartGeometry:

- prisPartGeomLabel
  - specifies a short text description of which may be used in conjunction with the geometric_rep_number of the geometric representation to uniquely identify a PrismaticPartGeometry
  - need not be specified for a particular PrismaticPartGeometry

- refPtXDim
  - specifies a numerical value for the position along the x-axis at which the cross-sectional geometry of a pseudo-prismatic part is defined
  - need not be specified for a particular PrismaticPartGeometry

- ppProfile
  - specifies a prismatic part profile associated with a PrismaticPartGeometry
  - need not be specified for a particular PrismaticPartGeometry
  - application assertion
    - each PrismaticPartGeometry is associated with one SectionProfile
    - each SectionProfile provides the cross-sectional geometry for zero, one or many PrismaticPartGeometry objects

- cardinalPoint
  - specifies a numerical identifier for one of 15 points which may be selected as the cardinal point of a prismatic section
  - is a point which retains the same position as the shape, dimensions, or orientation of a prismatic or pseudo-prismatic part is adjusted
  - need not be specified for a particular PrismaticPartGeometry

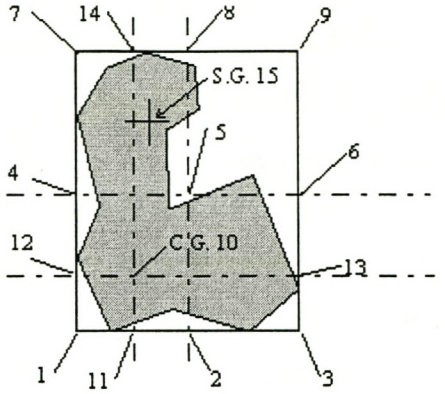


Figure A.2: Position of Possible Cardinal Points

8. PseudoPrismaticPart:

A pseudo prismatic part is a type of structural part and is defined by means of a series of 2 dimensional cross-sections along the length of the part. The geometry changes along its length and does so in a uniform fashion. The length defined in terms of "absolute" (measured from the origin of the coordinate system associated with the part) or "relative" (measured from some point along the x-axis of the coordinate system associated with the part, which is off-set by a given distance from the origin of the coordinate system). The origin of the coordinate system relating to a prismatic part lies at the centroid of the face at one end of the part and the x-axis follows the centroidal axis of the part. Each pseudo prismatic part is a tapered section.

Data associated with PseudoPrismaticPart:

- pppSourceRef
  - specifies a text description of a source where an explicit specification for a PseudoPrimaticPart is to be found
  - need not be specified for a particular PseudoPrismaticPart

- pppGeomAtPtX
  - specifies a prismatic part geometry for a PseudoPrismaticPart at a point along the x-axis
  - application assertion
    - each PseudoPrismaticPart is described by zero, one or many PrismaticPartGeometry objects
    - each PrismaticPartGeometry describes zero, one or many PseudoPrismaticPart objects

- pppLength
  - specifies a numerical value for the length of a PseudoPrismaticPart
  - need not be specified for a particular PseudoPrismaticPart

- pppLengthRelative
  - specifies whether or not the start of a given pseudo-prismatic part length is placed relative to the origin of the coordinate system associated with the part
  - need not be specified for a particular PseudoPrismaticPart

- pppLengthOffset
  - specifies a numerical value for the off-set applicable to a relative length
  - need not be specified for a particular PseudoPrismaticPart

- massPerLength
  - specifies a numerical value for the mass per unit length of a PseudoPrismaticPart
  - need not be specified for a particular PseudoPrismaticPart

- sectionClassification
  - specifies a numerical identifier for the classification of the PseudoPrismaticPart
  - need not be specified for a particular PseudoPrismaticPart

- pseudoPrisPartSelect
  - specifies the type of PseudoPrismaticPart as tapered

9. GeometricRepresentation:

Is a type of SteelStructuralFrameEntity that defines geometric characteristics of SPart and Element (10). A geometric representation is either a PrismaticPartGeometry or a SheetPartGeom.

Data associated with GeometricRepresentation:

- geometricRepNumber
  - specifies a unique numerical identifier for a GeometricRepresentaion

- geometryType
  - specifies the type of GeometricRepresentation as parametric

- geometricRepLabel
  - specifies a short text description that is used in conjunction with the geometric_rep_number to uniquely identify a GeometricRepresentation
  - need not be specified for a particular GeometricRepresentation

- geometrySource
  - specifies a text description of the source of the geometry
  - need not be specified for a particular GeometricRepresentation

- parametricGeomType
  - specifies the type of GeometricRepresentation as
    - sheet part geometry
    - prismatic part geometry

10. Element:

An element is a 1, 2, or 3 dimensional entity that is used to model a part of a structure during structural analysis (*AnalysisModel*).

An analysis model is a type of SteelStructuralFrameEntity that represents part of Structure (or Assembly) for structural analysis and is made up of Nodes (13) and Elements.

Data associated with Element:

- elementNumber
  - a unique numerical identifier for an element

- elementOrientation
  - a numerical value for the angle between the z-axis of an element and a line in the yz-plane that passes through the origin of the element's coordinate system, and meets the global z-axis of the structure
  - need not be specified for a particular Element

- elementMaterial
  - specifies a material for an element
  - need not be specified for a particular Element
  - application assertion
    - each Element has one Material
    - each Material is associated with zero, one or many Element objects

- elementGeometry
  - specifies a geometric representation for a Element
  - need not be specified for a particular Element
  - application assertion
    - each Element has one GeometricRepresentation
    - each GeometricRepresentation provides the geometry for zero, one or many Element objects

- parentModel
  - specifies an analysis model of which an Element is part
  - application assertion
    - each Element belongs to one AnalysisModel
    - each AnalysisModel contains one or many Element objects

- elementGroup
  - a group to which a element belongs
  - need not be specified for a particular Element
  - application assertion
    - each Element belongs to one ElementGroup
    - each ElementGroup contains one or many Element objects

- elementLabel
  - a short text description used in conjunction with the elementNumber to uniquely identify an Element
  - need not be specified for a particular Element

- dimensional_type
  - specifies the dimensionality of an Element as:
    - 1 dimensional
    - 2 dimensional
    - 3 dimensional
  - need not be specified for a particular Element

- pidNode1
  - persistent identifier for the starting node of the element

- eccentricity1
  - specifies an eccentricity associated with the starting Node of the Element
  - need not be specified for a particular Element
  - application assertion
    - each Element has one ElementEccentricity
    - each ElementEccentricity applies to zero, one or many EltNodeConnectivity objects

- fixity1
  - specifies a release associated with the starting Node of the Element
  - need not be specified for a particular Element
  - application assertion
    - associated with one Release
    - each Release associated with zero, one or many Element objects

- pidNode2
  - persistent identifier for the ending node of the element

- eccentricity2
  - specifies an eccentricity associated with the ending Node of the Element
  - need not be specified for a particular Element
  - application assertion
    - each Element has one ElementEccentricity
    - each ElementEccentricity applies to zero, one or many Element objects

- fixity2
  - specifies a release associated with the ending Node of the Element
  - need not be specified for a particular Element
  - application assertion
    - associated with one Release
    - each Release associated with zero, one or many Element objects

11. ElementEccentricity:

Element eccentricity is a type of SteelStructuralFrameEntity and is used when modeling Elements whose end-centers do not co-inside with their connecting Nodes (13).

Data associated with ElementEccentricity:

- xEccentricity
  - numerical value (in accordance with the specified units system) for the eccentricity of an element-end along the global y-axis of the structure
  - need not be specified for a particular ElementEccentricity

- yEccentricity
  - numerical value (in accordance with the specified units system) for the eccentricity of an element-end along the global x-axis of the structure
  - need not be specified for a particular ElementEccentricity

- zEccentricity
  - numerical value (in accordance with the specified units system) for the eccentricity of an element-end along the global z-axis of the structure
  - need not be specified for a particular ElementEccentricity

- elementEccentricityNumber
  - a unique numerical identifier for an Element

12. Release:

A release is a type of SteelStructuralFrameEntity that defines the degrees of freedom of an element at the point where it connects with a node.

Data associated with Release:

- releaseNumber
  - unique numeric identifier for a Release

- releaseLabel
  - a short text description that may be used in conjunction with a releaseNumber to uniquely identify a Release
  - need not be specified for a particular Release

- releaseDescription
  - a text description of a Release
  - need not be specified for a particular Release

- releaseAxialForce
  - a numerical value (in accordance with the specified units system) for the end-fixity in the linear x-direction of an element's coordinate system (i.e. the restraint against linear force)
  - a positive value indicates a linear spring
  - a value of zero indicates free displacement

- releaseYForce
  - a numerical value (in accordance with the specified units system) for the end-fixity in the linear y-direction of an element's coordinate system (i.e. the restraint against in-plane force)
  - a positive value indicates a linear spring
  - a value of zero indicates free displacement

- releaseZForce
  - a numerical value (in accordance with the specified units system) for the end-fixity in the linear z-direction of an element's coordinate system (i.e. the restraint against out-of-plane force)
  - a positive value indicates a linear spring
  - a value of zero indicates free displacement
  - the convention for representing a fixed displacement is a value of minus one (-1)

- releaseTorsionalMoment
  - a numerical value (in accordance with the specified units system) for the end-fixity about the x-axis of an element's coordinate system (i.e. the restraint against torsional bending)
  - a positive value indicates a rotational spring
  - a value of zero indicates free rotation

- releaseYBendingMoment
  - a numerical value (in accordance with the specified units system) for the end-fixity about the y-axis of an element's coordinate system (i.e. the restraint against major-axis bending)
  - a positive value indicates a rotational spring
  - a value of zero indicates free rotation

- releaseZBendingMoment
  - a numerical value (in accordance with the specified units system) for the end-fixity about the z-axis of an element's coordinate system (i.e. the restraint against minor-axis bending)
  - a positive value indicates a rotational spring
  - a value of zero indicates free rotation

13. Node:

A node is a theoretical point in space that is used in structural analysis to define a place where 2 or more Elements meet and where a Structure may be restrained.

Data associated with Node:

- nodeNumber
  - a unique numeric identifier for a Node

- nodeLabel
  - short text description, used in conjunction with nodeNumber to uniquely identify a Node
  - need not be specified for a particular Node

- nodeGrp
  - a group to which a Node belongs
  - need not be specified for a particular Node
  - application assertion
    - each Node belongs to one NodeGroup
    - each NodeGroup contains one or many Node objects

- parentModel
  - an analysis model to which a Node belongs
  - need not be specified for a particular Node
  - application assertion
    - Node is a component of one AnalysisModel
    - each AnalysisModel includes one or many Node objects

- restraints
  - boundary conditions for a Node
  - need not be specified for a particular Node
  - application assertion
    - Node is restrained by one BoundaryCondition
    - each BoundaryCondition restrains zero, one or many Node objects

- nodeCoords
  - a point at which a Node lies in the coordinate system of the parent analysis model
  - need not be specified for a particular Node
  - application assertion

- each Node is located by one Point
- each Point locates zero, one or many Node objects

14. Point :

A point is allocation in one, two, or three dimensional space defined by Cartesian coordinates (This has been adapted form the ISO 10303-42).

Data associated with Point:

- pointType
  - specifies the type of geometry used to define a Point as Cartesian or undefined

- xCoord
  - specifies a numerical value (in accordance with the specified units system) for the x-coordinate of a Point
  - need not be specified for a particular Point

- yCoord
  - specifies a numerical value (in accordance with the specified units system) for the y-coordinate of a Point
  - need not be specified for a particular Point

- zCoord
  - specifies a numerical value (in accordance with the specified units system) for the z-coordinate of a Point
  - need not be specified for a particular Point

15. CoordSystem:

A coordinate system defines a 3-axis Cartesian coordinate system by reference to a parent, or ultimately a global coordinate system. It also holds information relating to the application context of the coordinate system.

Data associated with CoordSystem:

- locatingCoordSys
  - specifies a(nother) CoordSystem that is used to locate a CoordSystem
  - need not be specified for a particular CoordSystem
  - application assertion
    - each CoordSystem is located by one CoordSystem
    - each CoordSystem locates zero, one or many CoordSystem objects

- csId
  - specifies a unique alpha-numeric identifier for a CoordSystem

- origin
  - specifies a position for a CoordSystem in terms of x, y and z coordinates of a parent CoordSystem
  - need not be specified for a particular CoordSystem
  - application assertion
    - each CoordSystem is located (at its origin) by one Point
    - each Point locates the origins of zero, one or many CoordSystem objects
  - in most situations, the parent Cartesian coordinate system at the top of the hierarchy (the coordinate system for the site) shall have its origin defined as (0,0,0).

- axisRotation
  - specifies a set of rotations about the x, y and z axes through which a CoordSystem is transformed
  - need not be specified for a particular CoordSystem
  - application assertion
    - each CoordSystem is rotated (about its x, y and z axes) by one Transformation
    - each Transformation rotates zero, one or many CoordSystem objects about their x, y and z axes

- csSignConvention
  - specifies a text description of the sign convention used in a CoordSystem
  - need not be specified for a particular CoordSystem
  - default assumption is that the z-axis is vertical and a right-handed coordinate system (with a positive value for rotation in a clockwise direction) is used

- coordSysType
  - specifies the type of CoordSystem as Cartesian

- coordSysUse
  - specifies the application context of a CoordSystem as joint system coordinate system, or part coordinate system, or feature coordinate system, or assembly coordinate system, or structure coordinate system, or site coordinate system, or undefined coordinate system

16. Transformation:

A transformation locates one coordinate system (CoordSystem) with respect to another. This part uses a right-handed Cartesian geometry with clockwise rotation taken as positive.

Data associated with Transformation:

- transformationId
  - specifies a unique alpha-numeric identifier for a Transformation

- transformationType
  - specifies the type of transformation as Cartesian
  - still to be extended with future versions of ISO 10303

- coeffX1
  - specifies a numerical value (in accordance with the specified units system) for the x-coordinate in the parent system of the terminus of a unit vector which begins at the origin of the child coordinate system and is coincident with the x-axis of that system

- coeffX2
  - specifies a numerical value (in accordance with the specified units system) for the x-coordinate in the parent system of the terminus of a unit vector which begins at the origin of the child coordinate system and is coincident with the y-axis of that system

- coeffX3
  - specifies a numerical value (in accordance with the specified units system) for the x-coordinate in the parent system of the terminus of a unit vector which begins at the origin of the child coordinate system and is coincident with the z-axis of that system

- coeffY1
  - specifies a numerical value (in accordance with the specified units system) for the y-coordinate in the parent system of the terminus of a unit vector which begins at the origin of the child coordinate system and is coincident with the x-axis of that system

- coeffY2
  - specifies a numerical value (in accordance with the specified units system) for the y-coordinate in the parent system of the terminus of a unit vector which begins at the origin of the child coordinate system and is coincident with the y-axis of that system

- coeffY3
  - specifies a numerical value (in accordance with the specified units system) for the y-coordinate in the parent system of the terminus of a unit vector which begins at the origin of the child coordinate system and is coincident with the z-axis of that system

- coeffZ1
  - specifies a numerical value (in accordance with the specified units system) for the z-coordinate in the parent system of the terminus of a unit vector which begins at the origin of the child coordinate system and is coincident with the x-axis of that system

- coeffZ2
  - specifies a numerical value (in accordance with the specified units system) for the z-coordinate in the parent system of the terminus of a unit vector which begins at the origin of the child coordinate system and is coincident with the y-axis of that system

- coeffZ3
  - z3 specifies a numerical value (in accordance with the specified units system) for the z-coordinate in the parent system of the terminus of a unit vector which begins at the origin of the child coordinate system and is coincident with the z-axis of that system

17. Gridline:

A gridline defines a line (of infinite length) that passes through two specified points on a horizontal plane. this line is used as a reference when locating structural members (StrucMember) and connections (StructConnection).

Data associated with Gridline:

- gridlineRef
  - specifies a unique numeric identifier for a Gridline

- refPoint1
  - specifies a set of coordinates for the first point through which a Gridline passes
  - application assertion
    - each Gridline is located (at its first reference point) by one Point
    - each Point locates (at their first reference points) zero, one or many Gridline objects

- refPoint2
  - specifies a set of coordinates for the second point through which a Gridline passes
  - application assertion
    - each Gridline is located (at its second reference point) by one Point
    - each Point locates (at their second reference point) zero, one or many Gridline objects
  - note that at present only Cartesian geometries are considered

- precedingGL
  - specifies a(nother) Gridline that precedes a Gridline in a grid
  - need not be specified for a particular Gridline
  - application assertion
    - each Gridline is preceded by one Gridline
    - each Gridline precedes zero, one or many Gridline objects

- distFromPrecedingGL
  - specifies a numerical value (in accordance with the specified unit system) for the distance between a Gridline and its preceding Gridline

- parentSet
  - specifies a set of Gridline objects to which this Gridline belongs
  - need not be specified for a particular Gridline
  - application assertion
    - each Gridline belongs to one GridlineSet
    - each GridlineSet contains one or many Gridline objects

- parentCoordSys
  - specifies a coordinate system to which a Gridline belongs
  - need not be specified for a particular Gridline
  - application assertion
    - each Gridline is located by one CoordSystem
    - each CoordSystem locates zero, one or many Gridline objects

18. GridlineSet:

A gridline set defines a set of uniform or non-uniform gridlines.

Data associated with GridlineSet:

- gridId
  - specifies a unique alpha-numeric identifier for a GridlineSet

- gridSequence
  - specifies a text description of the sequence of gridlines that make up a GridlineSet
  - need not be specified for a particular GridlineSet

- gridUse
  - specifies a text description of the purpose of a GridlineSet
  - need not be specified for a particular GridlineSet

19. BoundaryCondition:

A boundary condition is a type of SteelStructuralFrameEntity that holds information relating to 6 degrees of freedom, 3 translation and 3 rotation of a supporting node in an analysis model.

Data associated with BoundaryCondition:

- boundaryCondNumber
  - a unique numerical identifier for a BoundaryCondition

- BoundaryConditionLabel
  - short text description that may be used in conjunction with the boundary condition number to uniquely identify a BoundaryCondition
  - need not be specified for a particular BoundaryCondition

- boundaryCondDescription
  - specifies a text description of a boundary condition
  - need not be specified for a particular BoundaryCondition

- xSkewAngle
  - a numerical value (in accordance with the specified unit system) for the angel of rotation about a global x-axis of a structure when a BoundaryCondition applies to a skewed support
  - need not be specified for a particular BoundaryCondition

- ySkewAngle
  - a numerical value (in accordance with the specified unit system) for the angel of rotation about a global y-axis of a structure when a BoundaryCondition applies to a skewed support
  - need not be specified for a particular BoundaryCondition

- zSkewAngle
  - a numerical value (in accordance with the specified unit system) for the angel of rotation about a global z-axis of a structure when a BoundaryCondition applies to a skewed support
  - need not be specified for a particular BoundaryCondition

- bcXDisplacement
  - a numerical value (in accordance with the specified unit system) for the degree of freedom of displacement along the x-axis of a supporting node
  - positive value indicates a linear spring
  - a value of zero indicates free displacement

- need not be specified for a particular BoundaryCondition

- bcYDisplacement
  - a numerical value (in accordance with the specified unit system) for the degree of freedom of displacement along the y-axis of a supporting node
  - positive value indicates a linear spring
  - a value of zero indicates free displacement
  - need not be specified for a particular BoundaryCondition

- bcZDisplacement
  - a numerical value (in accordance with the specified unit system) for the degree of freedom of displacement along the z-axis of a supporting node
  - positive value indicates a linear spring
  - a value of zero indicates free displacement
  - need not be specified for a particular BoundaryCondition

- bcXRotation
  - a numerical value (in accordance with the specified unit system) for the degree of freedom of rotation about the x-axis of a supporting node
  - positive value indicates a rotating spring
  - a value of zero indicates free rotation
  - need not be specified for a particular BoundaryCondition

- bcYRotation
  - a numerical value (in accordance with the specified unit system) for the degree of freedom of rotation about the y-axis of a supporting node
  - positive value indicates a rotating spring
  - a value of zero indicates free rotation
  - need not be specified for a particular BoundaryCondition

- bcZRotation
  - a numerical value (in accordance with the specified unit system) for the degree of freedom of rotation about the z-axis of a supporting node
  - positive value indicates a rotating spring
  - a value of zero indicates free rotation
  - need not be specified for a particular BoundaryCondition

20. SectionProfile:

A section profile is a type of SteelStructuralFrameEntity and defines the cross-sectional geometry of a prismatic (or pseudo-prismatic) steel sections. It is either an AngleSect (22), ITypeSect (24), CircleSect (23), RectangleSect (25), TTypeSect (26).

Data associated with SectionProfile:

- sectionId
  - a unique alpha-numeric identifier for a SectionProfile

- profileShape
  - specifies the shape of a profile as
    - circle
    - rectangle
    - I-form
    - T-form
    - angle-form
  - need not be specified for a particular SectionProfile

- standardProfile
  - specifies the category of section profile as a
    - standard item
    - manufacturer's item
    - non-standard item

21. SectionProperties:

Section properties is a collection of static characteristics related to the cross-sectional geometry of a steel member.

Data associated with SectionProperties:

- inertiaMomentIy
  - specifies a numerical value (in accordance with the specified units system) for the bending moment of inertia (second moment of area) about the major (y) axis
  - need not be specified for a particular SectionProperties object

- inertiaMomentIz
  - specifies a numerical value (in accordance with the specified units system) for the bending moment of inertia (second moment of area) about the major (z) axis
  - need not be specified for a particular SectionProperties object

- profile
  - specifies a profile associated with SectionProperties object
  - application assertion
    - each SectionProperties object applies to one SectionProfile
    - each SectionProfile has zero, one or many SectionProperties

- torsionalConstantIx
  - specifies a numerical value (in accordance with the specified units system) for the polar moment of inertia (second moment of area) about the longitudinal (x) axis
  - need not be specified for a particular SectionProperties object

- sectionAreaAx
  - specifies a numerical value (in accordance with the specified units system) for the gross cross-sectional area of a section
  - need not be specified for a particular SectionProperties object

- shearAreaAy
  - specifies a numerical value (in accordance with the specified units system) for the effective shear area for the forces applied in the y-direction
  - need not be specified for a particular SectionProfile object

- shearAreaAz
  - specifies a numerical value (in accordance with the specified units system) for the effective shear area for the forces applied in the z-direction
  - need not be specified for a particular SectionProfile object

- radiusOfGyrationRy
  - specifies a numerical value (in accordance with the specified units system) for the radius of gyration about the major (y) axis
  - need not be specified for a particular SectionProperties object

- radiusOfGyrationRz
  - specifies a numerical value (in accordance with the specified units system) for the radius of gyration about the minor (z) axis
  - need not be specified for a particular SectionProperties object

- radiusOfGyrationRu
  - specifies a numerical value (in accordance with the specified units system) for the radius of gyration about the major principle (u) axis of a section with an asymmetrical profile
  - need not be specified for a particular SectionProperties object

- radiusOfGyrationRv
  - specifies a numerical value (in accordance with the specified units system) for the radius of gyration about the minor principle (u) axis of a section with an asymmetrical profile
  - need not be specified for a particular SectionProperties object

- surfacePerLength
  - specifies a numerical value (in accordance with the specified units system) for the surface area of a section for a given unit length
  - need not be specified for a particular SectionProperties object

- plasticModulusSy
  - specifies a numerical value (in accordance with the specified units system) for the plastic modulus about the major (y) axis
  - need not be specified for a particular SectionProperties object

- plasticModulusSz
  - specifies a numerical value (in accordance with the specified units system) for the plastic modulus about the minor (z) axis
  - need not be specified for a particular SectionProperties object

- warpingConstant
  - specifies a numerical value (in accordance with the specified units system) for the warping constant of a section
  - need not be specified for a particular SectionProperties object

- torsionalIndex
  - specifies a numerical value (in accordance with the specified units system) for the torsional index of a section
  - need not be specified for a particular Section-properties object

- bucklingParameter
  - specifies a numerical value (in accordance with the specified units system) for the buckling parameter of a section
  - need not be specified for a particular SectionProperties object

- horDistNaShearCentre
  - specifies a numerical value (in accordance with the specified units system) for the distance (measured along or parallel to the y-axis in the yz-plane) form the neutral axis to the shear center in a section with an asymmetrical profile
  - need not be specified for a particular SectionProperties object

- vertDistNaShearCentre
  - specifies a numerical value (in accordance with the specified units system) for the distance (measured along or parallel to the z-axis in the yz-plane) form the neutral axis to the shear center in a section with an asymmetrical profile
  - need not be specified for a particular SectionProperties object

- vertDistCentroidOrigin
  - specifies a numerical value (in accordance with the specified units system) for the distance (measured along or parallel to the y-axis in the yz-plane) from the geometrical centroid to the origin for the coordinate system used by a section with an asymmetrical profile
  - need not be specified for a particular SectionProperties object

- vertDistCentroidOrigin
  - specifies a numerical value (in accordance with the specified units system) for the distance (measured along or parallel to the z-axis in the yz-plane) from the geometrical centroid to the origin for the coordinate system used by a section with an asymmetrical profile
  - need not be specified for a particular SectionProperties object

- thetaAngleZAxisYAxis
  - specifies a numerical value (in accordance with the specified units system) for the plane angle of rotation (measured in a anti-clockwise direction) between the major (y) axis and the minor principle (v) axis of a section with an asymmetrical profile.
  - need not be specified for a particular SectionProperties object

- inertiaMomentIu
  - specifies a numerical value (in accordance with the specified units system) for the bending moment of inertia (or second moment of area) about the major principle (u) axis in a section with an asymmetrical profile
  - need not be specified for a particular SectionProperties object

- inertiaMomentIv
  - specifies a numerical value (in accordance with the specified units system) for the bending moment of inertia (or second moment of area) about the major principle (u) axis in a section with an asymmetrical profile
  - need not be specified for a particular SectionProperties object

- sectionModulusZyTop
  - specifies a numerical value (in accordance with the specified units system) for the elastic section modulus, about the major (y) axis, of the area of an asymmetrical profile that lies above the major axis (z increasing)
  - need not be specified for a particular SectionProperties object

- sectionModulusZyTop
  - specifies a numerical value (in accordance with the specified units system) for the elastic section modulus, about the major (y) axis, of the area of an asymmetrical profile that lies below the major axis (z decreasing)
  - need not be specified for a particular SectionProperties object

- sectionModulusZzLeft

- specifies a numerical value (in accordance with the specified units system) for the elastic section modulus, about the minor (z) axis, of the area of an asymmetrical profile that lies to the left of the minor axis (y decreasing)
- need not be specified for a particular SectionProperties object

- sectionModulusZzRight
  - specifies a numerical value (in accordance with the specified units system) for the elastic section modulus, about the minor (z) axis, of the area of an asymmetrical profile that lies to the right of the minor axis (y increasing)
  - need not be specified for a particular SectionProperties object

- sectionModulusZuVPos
  - specifies a numerical value (in accordance with the specified units system) for the elastic section modulus, about the major principle (u) axis, of the area of an asymmetrical profile that lies on the v increasing side of the major principle axis
  - need not be specified for a particular SectionProperties object

- sectionModulusZuVNeg
  - specifies a numerical value (in accordance with the specified units system) for the elastic section modulus, about the major principle (u) axis, of the area of an asymmetrical profile that lies on the v decreasing side of the major principle axis
  - need not be specified for a particular SectionProperties object

- sectionModulusZvUPos

  - specifies a numerical value (in accordance with the specified units system) for the elastic section modulus, about the minor principle (v) axis, of the area of an asymmetrical profile that lies on the u increasing side of the minor principle axis
  - need not be specified for a particular SectionProperties object

- sectionModulusZvUNeg

  - specifies a numerical value (in accordance with the specified units system) for the elastic section modulus, about the minor principle (v) axis, of the area of an asymmetrical profile that lies on the u decreasing side of the minor principle axis
  - need not be specified for a particular SectionProperties object

22. AngleSect:

An angle section is a type of SectionProfile that defines dimensions related to an angle section.

Data associated with AngleSect:

- depth
  - specifies a numerical value (in accordance with the specified units system) for the overall depth of an AngleSect

- thickness
  - specifies a numerical value (in accordance with the specified units system) for the mean leg thickness of an AngleSect

- width
  - specifies a numerical value (in accordance with the specified units system) for the overall width of an AngleSect
  - need not be specified for a particular AngleSect
  - if no width is specified, the section is assumed to be an equal angle (width = depth)

- intFilletRadius
  - specifies a numerical value (in accordance with the specified units system) for the internal fillet radius of an AngleSection
  - need not be specified for a particular AngleSect

- legSlope
  - specifies a numerical value (in accordance with the specified units system) for the slope of the leg of an angle section
  - need not be specified for a particular AngleSect

- edgeFilletRadius
  - specifies a numerical value (in accordance with the specified units system) for the edge fillet radius of an angle section
  - need not be specified for a particular AngleSect

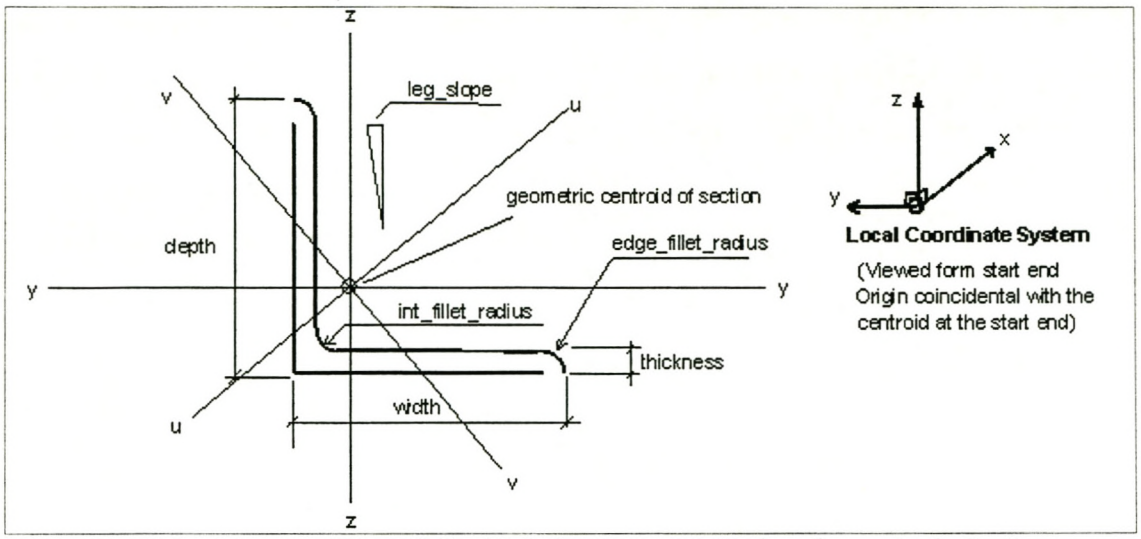


Figure A.3: Attributes necessary to describe an Angle Section

23. CircleSect:

A circular section is a type of SectionProfile that holds the dimensions of a solid or hollow circular sections.

Data associated with CircleSect:

- externalRadius
  - specifies a numerical value (in accordance with the specified units system) for the radius of a circular section, measured to the external surface in the case of a hollow section

- wallThickness
  - specifies a numerical value (in accordance with the specified units system) for the thickness of a wall of a hollow circular section
  - need not be specified for a particular CircleSect

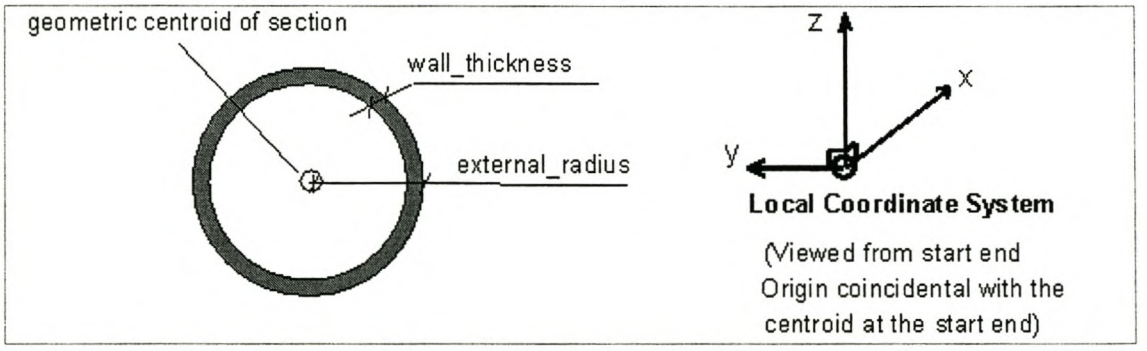


Figure A.4: Attributes necessary to describe a Circular Section

24. ITypeSection:

An I-type section is a type of SectionProfile that defines dimensions of a generic I type section. It inherits the coordinate system of section profile. The origin of the coordinate system coincides with the geometric centroid of the start end of the section. The x-axis of the coordinate system lies along the length of the section, the y-axis is perpendicular to the section web and the z-axis is perpendicular to the outer faces of the flanges. An I-type section may also be used to define the dimensions of a C (or U) type channel section. I type sections may be asymmetrical about the xy plane but are symmetrical about the xz-plane. C (op U) type sections are necessarily asymmetrical about the xz plane, and may be symmetrical about the xy-plane.

Data associated with ITypeSection:

- overallDepth
  - specifies a numerical value (in accordance with the specified units system) for the overall depth (upper face of top flange to lower face of bottom flange) of a section

- topFlangeWidth
  - specifies a numerical value (in accordance with the specified units system) for the overall breadth of the top flange of a section

- bottomFlangeWidth
  - specifies a numerical value (in accordance with the specified units system) for the overall breadth of the bottom flange of a section
  - need not be specified for a particular ITypeSect
  - if no value is given for the bottom flange width, the section is symmetrical about its xy plane, and the bottom flange width is equal to the top flange width

- topFlangeThickness
  - specifies a numerical value (in accordance with the specified units system) for the mean thickness of the top flange of a section

- btmFlangeThickness
  - specifies a numerical value (in accordance with the specified units system) for the mean thickness of the bottom flange of a section
  - need not be specified for a particular ITypeSect
  - if no value is given for the bottom flange thickness, the section is symmetrical about its xy plane, and the bottom flange thickness is equal to the top flange thickness

- webThickness
  - specifies a numerical value (in accordance with the specified units system) for the web of a section

- rootRadiusTop
  - specifies a numerical value (in accordance with the specified units system) for the fillet radius of the "root" between the web and the top flange of a section
  - need not be specified for a particular ITypeSect

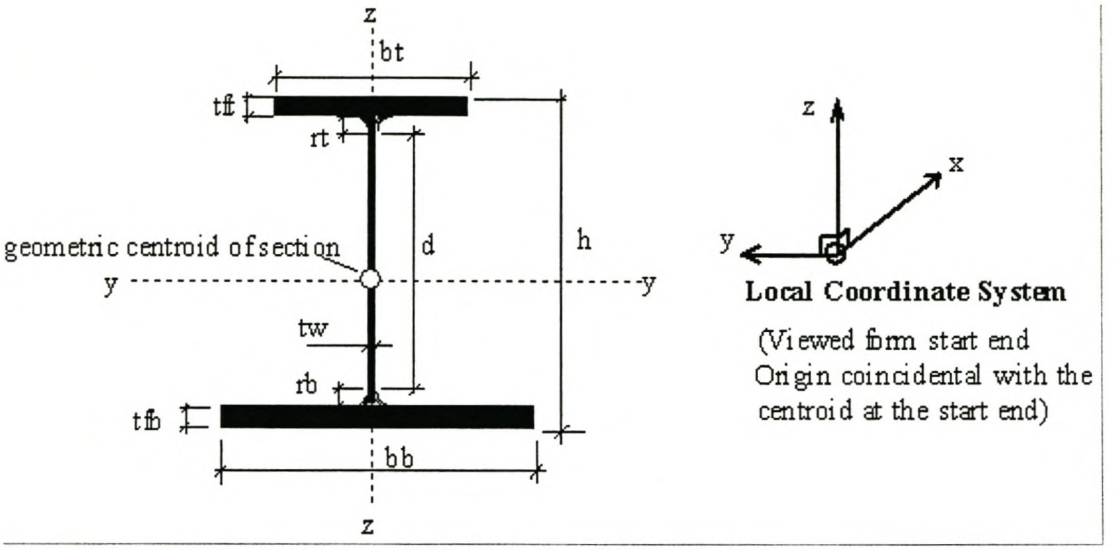
- rootRadiusBottom
  - specifies a numerical value (in accordance with the specified units system) for the fillet radius of the "root" between the web and the top flange of a section
  - need not be specified for a particular ITypeSect
  - if no value is given for the rootRadiusBottom and a value is provided for the top root radius, the section is symmetrical about its xy plane, and the bottom root radius is equal to the top root radius

- intDepth
  - specifies a numerical value (in accordance with the specified units system) for the internal depth (between fillets) of a section web
  - need not be specified for a particular ITypeSect

- topFlangeSlope
  - specifies a numerical value (in accordance with the specified units system) for the slope of the inner face of the top flange of a section
  - need not be specified for a particular ITypeSect

- btmFlangeSlope
  - specifies a numerical value (in accordance with the specified units system) for the slope of the inner face of the bottom flange of a section
  - need not be specified for a particular ITypeSect
  - if no value is given for the bottom flange slope and a value is provided for the top flange slope, the section is symmetrical about its xy plane, and the bottom flange slope is equal to the top flange slope

- edgeRadius
  - specifies a numerical value (in accordance with the specified units system) for the radius of the edge (or toe) chamfer applied to a flange
  - need not be specified for a particular ITypeSect
  - this data type may only be used when a section is symmetrical about its xy plane, and the bottom edge radius is equal to the top edge radius



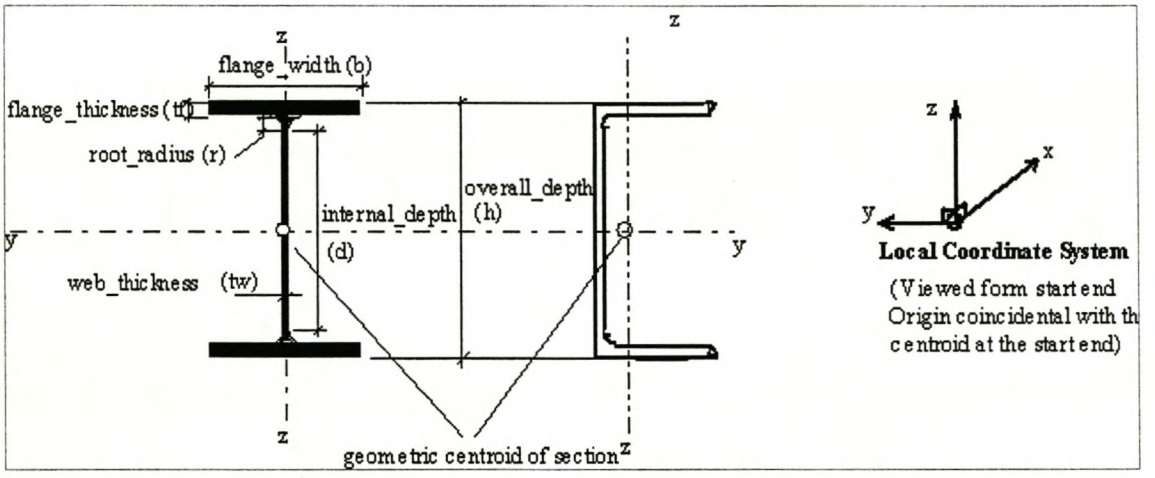


Figure A.5: Attributes necessary to describe an I-Section

25. RectAngleSect:

A rectangular section is a type of SectionProfile that has a rectangular cross-section.

Data associated with RectAngleSect:

- overallDepth
  - specifies a numerical value (in accordance with the specified units system) for the greater of the two (external face to external face) dimensions or (in the case of a square section) either of the two dimensions of a RectAngleSect measured in the z-axis

- overallWidth
  - specifies a numerical value (in accordance with the specified units system) for the lesser of the to (external face to external face) dimensions or (in the case of a square section) either of the two dimensions of a RectAngleSect measured in the y-axis
  - need not be specified for a particular RectAngleSect
  - if no value is provided for the overall width, the section is assumed to be square

- wallThickness
  - specifies a numerical value (in accordance with the specified units system) for the thickness of the wall of a hollow RectAngleSect
  - need not be specified for a particular RectAngleSect
  - if no value is provided for the wall thickness, the section is assumed to be solid

- intFilletRadius
  - specifies a numerical value (in accordance with the specified units system) for the radius of the fillet of a hollow section's internal corner
  - need not be specified for a particular RectAngleSect

- extFilletRadius
  - specifies a numerical value (in accordance with the specified units system) for the radius of the external corner of a hollow or solid section
  - need not be specified for a particular RectAngleSect

26. TTypeSect:

A T-type section is a type of SectionProfile that defines the dimensions of a generic T-type section. It inherits the coordinate system of SectionProfile. The origin of the coordinate system is coincides with the geometric centroid of the start end of the member. The x-axis of the coordinate system is perpendicular to the (midline of the) member web, the z-axis is perpendicular to the outer face of the flange. T-type sections are symmetrical about the xz-plane and asymmetrical about the yz-plane.

Data associated with TTypeSect:

- overallDepth
  - specifies a numerical value (in accordance with the specified units system) for the overall depth (upper face of flange to lower tip of web) of a section

- flangeWidth
  - specifies a numerical value (in accordance with the specified units system) for the overall breadth of the flange of a section

- flangeThickness
  - specifies a numerical value (in accordance with the specified units system and appropriate standard) for the thickness of the flange of a section

- webThickness
  - specifies a numerical value (in accordance with the specified units system) for the thickness of the web of a section

- rootRadius
  - specifies a numerical value (in accordance with the specified units system) for the fillet radius of the "root" between the web and the flange of a section
  - need not be specified for a particular TTypeSect

- flangeSlope
  - specifies a numerical value (in accordance with the specified units system and the appropriate standard) for the angle of slope of the inner face of the flange
  - need not be specified for a particular TTypeSect

- webSlope
  - specifies a numerical value (in accordance with the specified units system and the appropriate standard) for the angle of slope of the face of the web
  - need not be specified for a particular TTypeSect

27. SFeature:

A structural feature is a type of SteelStructuralFrameEntity that defines the geometry of LocatedFeature in terms of the material removed from the part. A structural feature can either be a CuttingPlane (28), EdgeChamfer (29), PrismaticEndFeature (30), ProceduralFeat (34).

Data associated with SFeature:

- sFeatureId
  - specifies a unique alpha-numeric identifier for an SFeature

- sFeatureType
  - specifies the type of feature as
    - cutting plane
    - edge chamfer
    - prismatic end feature
    - procedural feature

28. Cutting_plane:

A cutting plane is a type of structural feature and defines a boundary between a part, LocatedPart and material that is to be removed from a part. The feature is rotated, in relation to the part, so that the xy-plane of its coordinate system lines up with the cut required. Waste material is removed from the x-positive side of the plane.

29. EdgeChamfer:

An edge chamfer is a type of structural feature and describes the geometry and location of a chamfer with respect to the edge of a part. An edge chamfer has its own coordinate system. The length and orientation of an edge chamfer cut are defined in the yz-plane of the coordinate system. The cutting direction is parallel to the x-axis of the coordinate system. The orientation of an edge chamfer is described in relation to the original part-edge that is to be chamfered and one of the surfaces of a part that meets the edge to be chamfered, the reference surface. The x-axis of the edge chamfer lies along the original edge of the part, the xy-plane of the edge chamfer coordinate system lies on the reference surface and the z-axis is orthogonal to the reference surface.

- original edge that is a right angle
  - z-axis lies an the part-surface adjacent to the reference surface
- during the cut
  - EdgeChamfer moves around the reference surface in a clockwise fashion

Data associated with EdgeChamfer:

- followRound
  - specifies whether or not the EdgeChamfer is applied along the complete edge of the part

- edgeChamfDepth
  - specifies a numerical value for the depth of an edge chamfer along the positive z-axis
  - if edge chamfer is applied to a right angled edge, the edge edgeChamfDepth is the distance between the original edge of a part and the cutting line on the adjacent surface of that part
  - if edge chamfer is applied to an edge that is not a rectangle, the edgeChamfDepth is the distance from any point on the original edge of the part to a point on the plane of the cut measured along a line that is orthogonal to the reference surface
  - need not be specified for a particular EdgeChamfer

- edgeChamfWidth
  - specifies a numerical value for the width of an edge chamfer measured along the positive y-axis
  - it's the distance between the original edge of a part and the cutting line on the reference surface of that part
  - need not be specified for a particular EdgeChamfer

- edgeChamferRadius
  - specifies a numerical value for the radius of a concave curved chamfer applied to an edge
  - need not be specified for a particular EdgeChamfer

- edgeRoundingRadius
  - specifies a numerical value for the radius of a convex curved chamfer applied to an edge
  - need not be specified for a particular EdgeChamfer

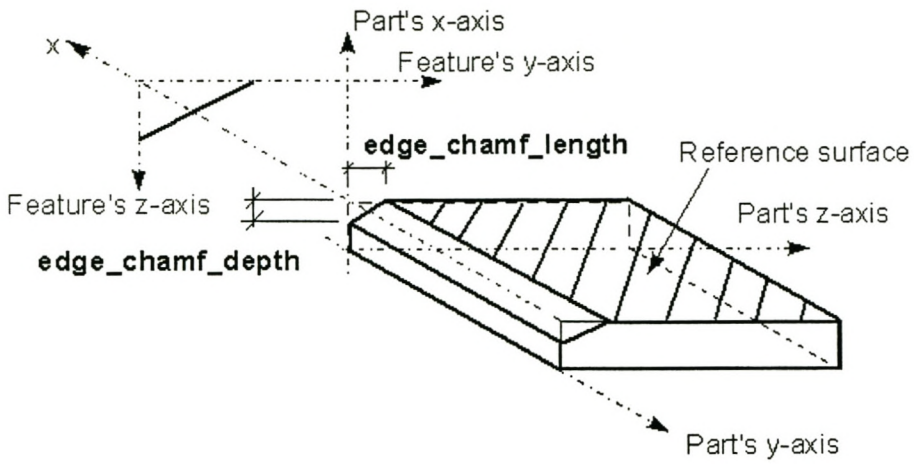


Figure A.6: Attributes of an Edge Chamfer

30. PrismaticEndFeature:

A PrismaticEndFeature is a type of SFeature that defines the geometry and location of a family of features commonly applied to prismatic parts. In relation to the coordinate system associated with a prismatic part, a prismatic end feature is placed either at the start face (x = 0,0) and at the top (z = maximum positive value), or at the end face (x = maximum positive value) and at the bottom (z = maximum negative value). A PrismaticEndFeature can be either a Chamfer (31), Notch (32) or a SkewedEnd (33).

Data associated with PrismaticEndFeature:

- cuttingMethod
  - specifies the method used to create the feature as
    - sawn
    - flame cut
    - sheared
    - punched
    - drilled
  - need not be specified for a particular PrismaticEndFeature

- topOrBtmEdge
  - specifies the prismatic part-edge to which a PrismaticEndFeature is applied
    - top
    - bottom
  - need not be specified for a particular PrismaticEndFeature

- startOrEnd
  - specifies the prismatic part-end at which a PrismaticEndFeature is applied
    - start
    - end

- prismaticEndFeatureType
  - specifies the type of PrismaticEndFeature as a
    - notch
    - chamfer
    - skewed end

31. Chamfer:

A Chamfer is a type of PrismaticEndFeature and defines a plane cut at the end of a prismatic part. It has its own coordinate system. The length and orientation of a chamfer cut are defined in the yz-plane of the coordinate system. The cutting direction is parallel to the x-axis of the coordinate system and is applied at the start or end face of a prismatic part. In the case of a prismatic part with a SkewedEnd (33) the chamfer applied either to original (square) face or the new face created by the skew cut.

The x-axis of the chamfer coordinate system lies along an edge of the relevant face. The y-axis of the chamfer coordinate system is orthogonal to the face. The z-axis of the chamfer coordinate system lies in the same plane as the face. The depth of a chamfer, measured along the z-axis, is less than the depth of the prismatic part. Plane cuts applied to an original face, which had a depth equal or greater than the part to which they applied, would produce a skewed end.

Data associated with Chamfer:

- chamfDepth
  - specifies a numerical value for the depth of a chamfer along the z-axis

- chamfLength
  - specifies a numerical value for the length of a chamfer along the y-axis

- originalFace
  - specifies whether or not a chamfer is applied to an original face or to a residual face created after the end of a prismatic part has been cut skew

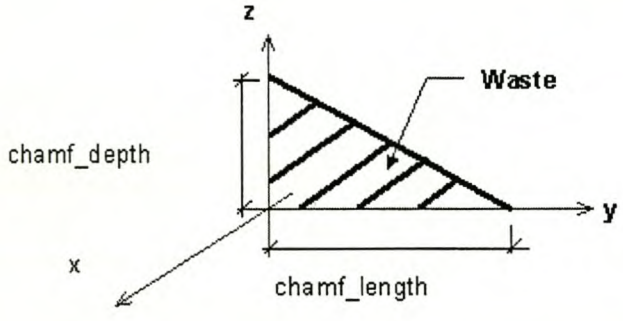


Figure A.7: Attributes of a Chamfer

32. Notch:

A Notch is a type of PrismaticEndFeature that defines geometry and location of a notch with respect to the end of a prismatic part. A Notch has it's own coordinate system. The area of material removed by a notch is defined in the yz-plane of the coordinate system. The cutting direction is given by the x-axis of the coordinate system. A Notch is applied to the start or and of a prismatic part.

In the case of a prismatic part with a skewed end a Notch is applied either to the original (square) end or the new end created by the skew cut. The x-axis of the Notch coordinate system lies along the top or bottom edge of the end and the xy-plane of the Notch is parallel to the xy-plane of the prismatic part.

The coordinate system of the Notch remains static with respect to the coordinate system of the prismatic part during a cut, thus the Notch can only be cut along a straight line

Data associated with Notch:

- notchLength
  - specifies a numerical value for the length of a notch measured along the y-axis

- notchDepth
  - specifies a numerical value for the depth of a notch measured along the z-axis

- notchRadius
  - specifies a numerical value for the radius of the corner fillet of the notch
  - need not be specified for a particular Notch

- originalFace
  - specifies whether or not a Notch is applied to the original (square) end of a prismatic part

33. SkewedEnd:

A SkewedEnd is a type of PrismaticEndFeature that defines location and orientation of a cutting plane used to cut the end of a prismatic part or a pseudo-prismatic part.

Data associated with SkewedEnd:

- ySkewAngle
  - specifies a numerical value for the angle between the yz-plane of the cut and the yz-plane of the part as measured in the xy-plane of the part.
  - need not be specified for a particular SkewedEnd.

- zSkewAngle
  - specifies a numerical value for the angle between the yz-plane of the cut and the yz-plane of the part as measured in the xz-plane of the part.
  - need not be specified for a particular SkewedEnd.

34. ProceduralFeat:

A ProceduralFeat is a type of SFeature that defines a relationship between a Hole (35) and a Layout (41). It places a Hole at a point in a Layout.

Data associated with ProceduralFeat:

- holes
  - specifies a hole associated with a ProceduralFeat
  - need not be specified for a particular ProceduralFeat
  - application assertion
    - each ProceduralFeat is associated with one Hole
    - each Hole is associated with zero, one or many ProceduralFeat objects

- pfUse
  - specifies a text description of a ProceduralFeat
  - need not be specified for a particular ProceduralFeat

- pfArrangement
  - specifies a layout associated with a ProceduralFeat
  - need not be specified for a particular ProceduralFeat
  - application assertion
    - each ProceduralFeat is associated with one Layout
    - each Layout positions zero, one or many ProceduralFeat objects

35. Hole:

A Hole is a specialized feature that has a closed cutting edge and results in the creation of purely internal waste. It is defined in the yz-plane of a feature's coordinate system and may be oval or circular. The cutting direction is defined as lying parallel to the x-axis.

Data associated with Hole:

- holeRadius
  - specifies a numerical value for the radius of a Hole or for the radii of the semi-circular areas at the two ends of an oval slot

- slotLength
  - specifies a numerical value for the distance between the centres of the semi-circular areas at he two ends of an oval slot
  - need not be specified for a particular Hole

- holeType
  - specifies the type of Hole as
    - circular
    - slotted
    - undefined

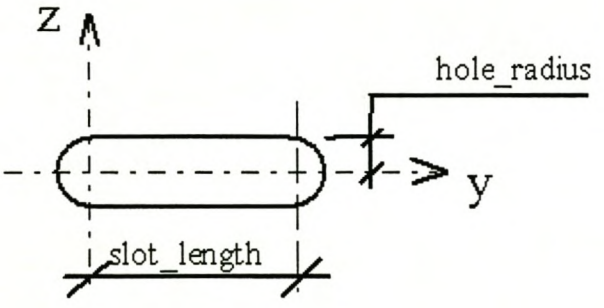


Figure A.8: Attributes of a Hole

36. LocatedPart:

LocatedPart is an SPart that has been located and oriented relative to a manufacturing assembly (ManufactAsbly) and thus to a Structure.

Data associated with LocatedPart:

- parentMa
  - specifies a manufacturing assembly to which a LocatedPart belongs
  - application assertion
    - each LocatedPart belongs to one ManufactAsbly
    - each ManufactAsbly describes zero, one or many LocatedPart objects

- shippingRef
  - specifies a alpha-numeric identifier for a LocatedPart to be used during the erection process
  - need not be specified for a particular LocatedPart

- locPartCoordSys
  - specifies a coordinate system for a LocatedPart
  - need not be specified for a particular LocatedPart
  - application assertion
    - each LocatedPart is located by one CoordSystem
    - each CoordSystem locates zero, one or many LocatedPart objects

- descriptivePart
  - specifies a specific part that is located by a LocatedPart
  - application assertion
    - each LocatedPart is described by one SPart
    - each SPart describes zero, one or many LocatedPart objects

- workshopRef
  - specifies an alpha-numeric identifier for a LocatedPart to be used during fabrication
  - need not be specified for a particular LocatedPart

- LocatedPartId
  - specifies a unique alpha-numeric identifier for a LocatedPart

- designRef
  - specifies an alpha-numeric identifier for a LocatedPart to be used during the design process

- totalMass
  - specifies a numerical value for the total mass of a LocatedPart
  - need not be specified for a particular LocatedPart

- partsGrp
  - specifies a group to which a LocatedPart belongs
  - need not be specified for a particular LocatedPart
  - application assertion
    - each LocatedPart belongs to one L_parts_group
    - each L_parts_group contains one or many LocatedPart objects

- locPartNature
  - specifies a text description of the role of the LocatedPart with respect to the connections of the structure
  - need not be specified for a particular LocatedPart

- locPartDescription
  - specifies a text description of a LocatedPart
  - need not be specified for a particular LocatedPart

37. PartJoint:

A PartJoint is a logical connection between a pair of located parts.

Data associated with PartJoint:

- partJointId
  - specifies a unique alpha-numeric identifier for a PartJoint

- requiredJoints
  - specifies a located joint system associated with a PartJoint
  - application assertion
    - each PartJoint requires one or many LocatedJointSystem objects
    - each LocatedJointSystem physically fulfils zero, one or many Part_joint objects

- logicallyJoinedParts
  - specifies a located part associated with a PartJoint
  - application assertion
    - each PartJoint (logically) connects two LocatedPart objects
    - each LocatedPart is (logically) connected (to another part) by zero, one or many PartJoint objects

- pjNature
  - specifies a text description of a PartJoint
  - need not be specified for a particular PartJoint

38. LocatedJointSystem:

A LocatedJointSystem is an SjointSystem (44) that has been located and oriented to a manufacturing assembly and thus to a Structure. The located parts involved in a part joint are physically connected by the application of one or more located joint systems. A located system requires the application of one or more joint dependant features to the connected located part.

Data associated with LocatedJointSystem:

- parentMa
  - specifies a manufacturing assembly to which a LocatedJointSystem belongs
  - application assertion
    - each LocatedJointSystem belongs to one ManufactAsbly
    - each ManufactAsbly requires zero, one or many LocatedJointSystem objects

- locJointSysId
  - specifies a unique alpha-numeric identifier for a LocatedJointSystem

- locJntSysCoordSys
  - specifies the coordinate system for a LocatedJointSystem
  - need not be specified for a particular LocatedJointSystem
  - application assertion
    - each LocatedJointSystem is located by one CoordSystem
    - each CoordSystem locates zero, one or many LocatedJointSystem objects

- descriptiveJtSys
  - specifies a specific joint system that is located by a LocatedJointSystem
  - application assertion
    - each LocatedJointSystem is described by one S_joint_system
    - each S_joint_system describes zero, one or many LocatedJointSystem objects

- placeOfAssembly
  - specifies the place of assembly as site or workshop
  - need not be specified for a particular LocatedJointSystem

- jointsGrp
  - specifies a group to which a LocatedJointSystem belongs
  - need not be specified for a particular LocatedJointSystem
  - application assertion

- each LocatedJointSystem belongs to a L_joint_systems_group
- each L_joint_systems_group contains one or many LocatedJointSystem objects

39. LocatedFeature:

A LocatedFeature is a JointDepFeature (40).

Data associated with LocatedFeature:

- modifiedPart
  - specifies a located part to which a located feature applies
  - application assertion
    - each LocatedFeature modifies one LocatedPart.
    - each LocatedPart is modified by zero, one or many LocatedFeature objects

- locFeatId
  - specifies a unique alpha-numeric identifier for a LocatedFeature

- locFeatCoordSys
  - specifies a coordinate system for a LocatedFeature
  - need not be specified for a particular LocatedFeature
  - application assertion
    - each LocatedFeature is located by one CoordSystem
    - each CoordSystem locates zero, one or many LocatedFeature objects.

- descriptiveFeaure
  - specifies a specific feature for a LocatedFeature
  - application assertion
    - each LocatedFeature is described by one Sfeature
    - each SFeature describes zero, one or many LocatedFeature objects

- locFeatDescription
  - is a text description of a LocatedFeature
  - need not be specified for a particular LocatedFeature

- locFeatUse
  - specifies the LocatedFeature as joint dependent or joint independent

40. JointDepFeature:

A JointDepFeature is a type of LocatedFeature that provides an association between a located feature and a located joint system in situations where a feature is created as part of a joint.

Data associated with JointDepFeature:

- jtDepFeatDescr
  - specifies a text description of a JointDepFeature
  - need not be specified for a particular JointDepFeature

- featureForJoint
  - specifies a located joint system that makes use of a JointDepFeature
  - application assertion
    - each JointDepFeature is associated with one LocatedJointSystem
    - each LocatedJointSystem has zero, one or many JointDepFeature objects

41. Layout:

A Layout is a set of reference points lying in a plane. The coordinate system is that of a located joint system and the points defined in a layout lie in the yz plane of that system.

Data associated with Layout:

- layoutPoints
  - specifies a set of two 2D coordinates for a point in a layout
  - application assertion
    - each Layout has its origin located by one Point
    - each Layout is defined by one or many Point objects
  - the first point of a Layout is 0,0 – therefore the origin of the yz-plane – and is selected to provide y and z values for the remaining points in the layout

- layoutOrigin
  - specifies a set of 2D coordinates for the first point of a Layout – thus providing an offset for the set of Layout points
  - need not be specified for a particular Layout

- layoutDescription
  - specifies a text description of a Layout
  - need not be specified for a particular Layout

42. LocatedPart:

A located part is a SPart that has been located and oriented relative to a *ManufactAssembly* and thus to a Structure.

A ManufactAsbly is a type of Assembly that defines a subset of a Structure. A ManufactAsbly is any node in the composition hierarchy created during manufacture and erection - the top node of this hierarchy corresponding to the complete structure. Manufacturing assemblies corresponding to nodes below the top node are brought together with LocatedJointSystems and LocatedPart to form higher level assemblies. The composition of a physical structure need not correspond to the composition of a designed structure.

Data associated with LocatedPart:

- parentMa
  - specifies a manufacturing assembly to which a LocatedPart belongs
  - application assertion
    - each LocatedPart belongs to one ManufactAsbly
    - each ManufactAsbly describes zero, one or many LocatedPart objects

- shippingRef
  - specifies a alpha-numeric identifier for a LocatedPart to be used during the erection process
  - need not be specified for a particular LocatedPart

- locPartCoordSys
  - specifies a coordinate system for a LocatedPart
  - need not be specified for a particular LocatedPart
  - application assertion
    - each LocatedPart is located by one CoordSystem
    - each CoordSystem locates zero, one or many LocatedPart objects

- descriptivePart
  - specifies a specific part that is located by a LocatedPart
  - application assertion
    - each LocatedPart is described by one SPart
    - each SPart describes zero, one or many LocatedPart objects

- workshopRef
  - specifies an alpha-numeric identifier for a LocatedPart to be used during fabrication
  - need not be specified for a particular LocatedPart

- LocatedPartId
  - specifies a unique alpha-numeric identifier for a LocatedPart

- designRef
  - specifies an alpha-numeric identifier for a LocatedPart to be used during the design process

- totalMass
  - specifies a numerical value for the total mass of a LocatedPart
  - need not be specified for a particular LocatedPart

- partsGrp
  - specifies a group to which a LocatedPart belongs
  - need not be specified for a particular LocatedPart
  - application assertion
    - each LocatedPart belongs to one L_parts_group
    - each L_parts_group contains one or many LocatedPart objects

- locPartNature
  - specifies a text description of the role of the LocatedPart with respect to the connections of the structure
  - need not be specified for a particular LocatedPart

- locPartDescription
  - specifies a text description of a LocatedPart
  - need not be specified for a particular LocatedPart

43. Connector:

A connector is a type of SteelStructuralFrameEntity. It holds information relating to the design of welded or bolted connections.

Data associated with Connector:

- connectorNumber
  - specifies a unique numeric identifier for a Connector

- connectorSpec
  - specifies physical and geometrical characteristics of a Connector
  - need not be specified for a particular Connector
  - application assertion
    - each Connector is described by one SJointSystem
    - each SJointSystem describes zero, one or many Connector objects

- connectorType
  - specifies the connection assumed appropriate for the design, as
    - pinned
    - or semi-rigid full strength
    - or semi-rigid partial strength
    - or rigid partial strength
    - states a basic assumption used during analysis and design, and may be used as an initializer by some applications
    - need not be specified for a particular Connector

- connectorLabel
  - specifies a short text description used in conjunction with the connector_number to uniquely identify a Connector
  - need not be specified for a particular Connector

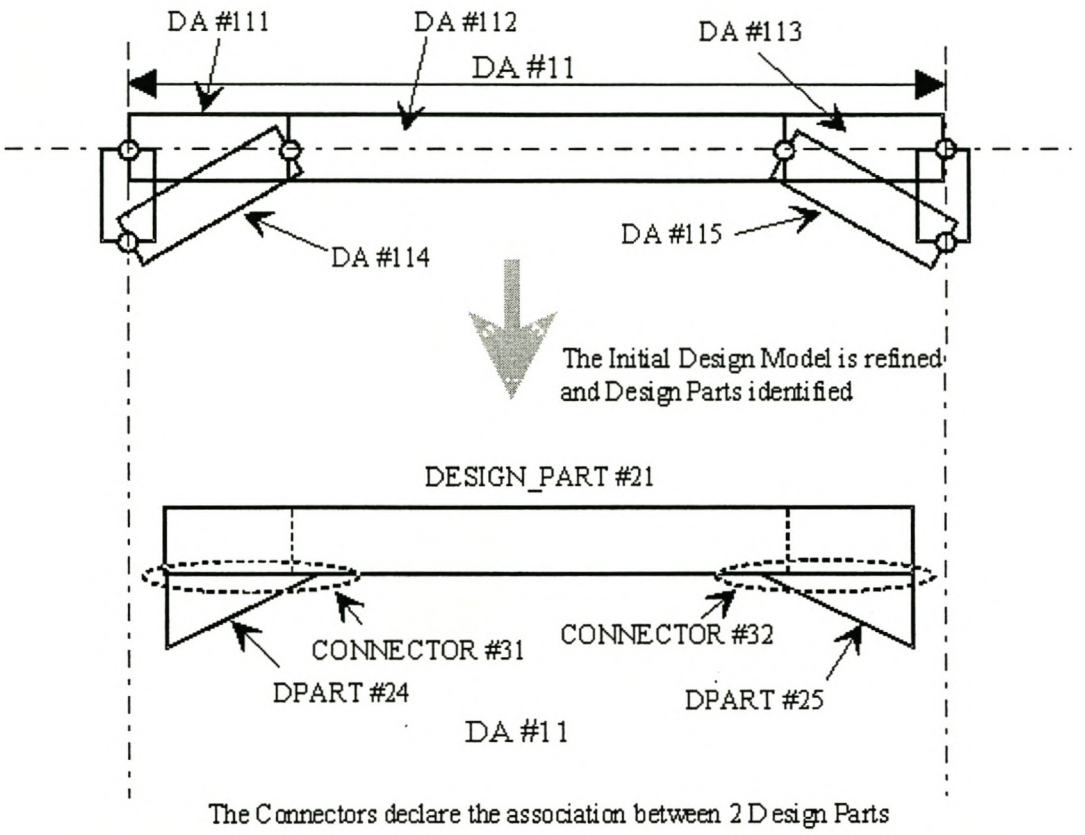
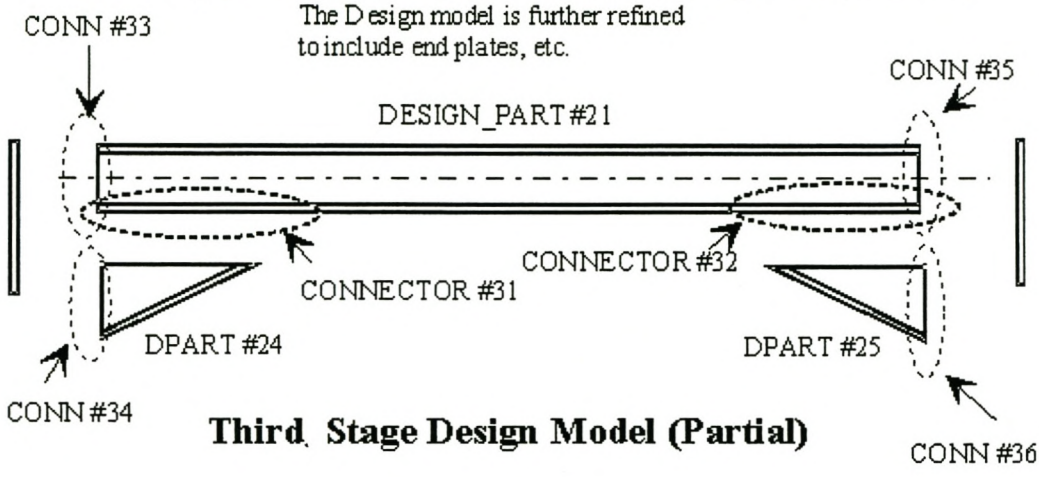
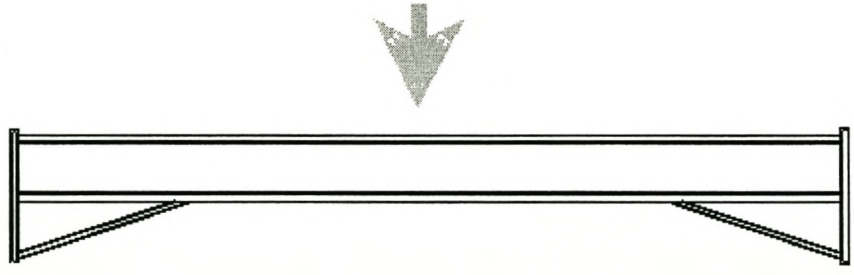


Figure A.8: Describing the Connections of Design Parts

Figure A.9: Describing the Connections of Design Parts (Continues)





44. SjointSystem:

Structural joint system is a type of SteelStructuralFrameEntity that defines the configuration of LocatedJointSystem. It can be either a BoltSystem (45), or a WeldSystem (50).

Data associated with SJointSystem:

- sJointSystemId
  - specifies a unique alpha-numerical identifier for a SJointSystem

- sJointSystemType
  - specifies the type of specific joint system as a
    - bolt system, or
    - weld system

45. BoltSystem:

A bolt system is a type of Structural joint system. It associates one or more tokens of a particular BoltMechanism (46) with a pattern or Layout.

A Layout is a set of reference points lying in a plane. The coordinate system is that of a LocatedJointSystem and the points defined in a layout lie in the yz plane of that system.

Data associated with BoltSystem:

- bsLayout
  - specifies the layout of a BoltSystem
  - may not be specified for a particular BoltSystem
  - application assertion
    - each BoltSystem is arranged in accordance with one Layout
    - each Layout specifies the arrangement of zero, one or many BoltSystem objects

- bsNumber
  - specifies a unique numeric identifier for a BoltSystem
  - need not be specified for a particular BoltSystem

- bmSpec
  - specifies a bolt mechanism used in a BoltSystem
  - need not be specified for a particular BoltSystem
  - application assertion
    - each BoltSystem uses one BoltMechanism
    - each BoltMechanism is used in one or many BoltSystem objects

46. BoltMechanism:

A bolt mechanism is a set of items that includes a Bolt (47) and, typically, includes a Washer (49) and a Nut (48)

Data associated with BoltMechanism :

- bmId
  - specifies a unique alpha-numeric identifier for a BoltMechanism

- bmSequence
  - specifies a text description of a sequence of washers, nuts, and parts placed on a bolt in a BoltMechanism

- nuts
  - specifies a nut used in a BoltMechanism
  - need not be specified for a particular BoltMechanism
  - application assertion
    - each BoltMechanism uses one or many Nut objects
    - each Nut is used in zero, one or many BoltMechanism objects

- washers
  - specifies a washer used in a BoltMechanism
  - need not be specified for a particular BoltMechanism
  - application assertion
    - each BoltMechanism uses one or many Washer objects
    - each Washer is used in zero, one or many BoltMechanism objects

- boltSpec
  - specifies a bolt used in a BoltMechanism
  - application assertion
    - each BoltMechanism uses one Bolt
    - each Bolt is used in zero, one or many BoltMechanism objects

47. Bolt:

A bolt is a type of SteelStructuralFrameEntity that provides a description of a bolt type, in terms of dimensions and other specifications, and that can form part (perhaps the only part) of one or more BoltMechanism.

Data associated with Bolt:

- boltId
  - specifies a unique alpha-numaric identifier for a Bolt

- boltType
  - specifies the type of bolt as slip or friction grip
  - need not be specified for a particular Bolt

- standardBolt
  - specifies the category of Bolt as standard item, or manufacturers item, or non-standard item

- boltHeadShape
  - specifies a text description of the shape of a bolt head
  - need not be specified for a particular Bolt

- lengthOfShank
  - specifies a numerical measurement (in accordance with the appropriate standard and the specified units system) for the length of the shank of a Bolt
  - need not be specified for a particular Bolt

- nominalLength
  - specifies a numerical measurement (in accordance with the appropriate standard and the specified units system) for the nominal length of a Bolt
  - need not be specified for a particular Bolt

- boltDiameter
  - specifies a numerical measurement (in accordance with the appropriate standard and the specified units system) for the nominal diameter of a Bolt
  - need not be specified for a particular Bolt

- boltStrength
  - specifies the material strength of a Bolt as normal, or high strength, or special
  - need not be specified for a particular Bolt

- boltGrade
  - specifies a text description (in accordance with the appropriate standard) for the grade of a Bolt
  - need not be specified for a particular Bolt

- boltPreload
  - specifies a numerical measurement (in accordance with the appropriate standard and the specified units system) for the pre-load applied to a Bolt
  - need not be specified for a particular Bolt

- boltMaterial
  - specifies a text description of the material from which a Bolt is manufactured
  - need not be specified for a particular Bolt

- distAcrVertices
  - specifies a numerical measurement (in accordance with the appropriate standard and the specified units system) for the distance between opposite vertices of the head of a Bolt
  - need not be specified for a particular Bolt

- distAcrFlats
  - specifies a numerical measurement (in accordance with the appropriate standard and the specified units system) for the distance between opposite flats of the head of a Bolt
  - need not be specified for a particular Bolt

- reducedSectionArea
  - specifies a numerical measurement (in accordance with the appropriate standard and the specified units system) for the reduced section area of the threaded part of the shaft of a Bolt
  - need not be specified for a particular Bolt

- fullSectionArea
  - specifies a numerical measurement (in accordance with the appropriate standard and the specified units system) for the full sectional area of the shaft of a Bolt
  - need not be specified for a particular Bolt

48. Nut:

A nut is a type of SteelStructuralFrameEntity that provides a description of a nut type, in terms of dimensions and other specifications and that can from part of one or more BoltMechanisms.

Data associated with Nut:

- nutId
  - specifies a unique alpha-numeric identifier for a Nut

- standardNut
  - specifies the category of Nut as
    - standard item
    - manufacture's item
    - or non-standard item

- boltDiameter
  - specifies a numerical value (in accordance with the specified units system and the appropriate standard) for the nominal diameter of a bolt that would fit a particular Nut

- nutShape
  - specifies a text description of the shape of a Nut
  - need not be specified for a particular Nut

- nutDepth
  - specifies a numerical value (in accordance with the specified units system) for the depth of a Nut (i.e. distance between two faces of a nut)
  - need not be specified for a particular Nut

- nutGrade
  - specifies a short text description (in accordance with the appropriate standard) for the grade of a Nut
  - need not be specified for a particular Nut

- nutStrength
  - specifies the material strength of a Nut as
    - normal
    - high strength
    - or special
  - need not be specified for a particular Nut

- nutMaterial
  - specifies a short text description of the material from which a Nut is manufactured
  - need not be specified for a particular Nut

- distAcrVertices
  - specifies a numerical value (in accordance with the specified units system and the appropriate standard) for the distance between opposite vertices of a Nut
  - need not be specified for a particular Nut
- distAcrFlats
  - specifies a numerical value (in accordance with the specified units system and the appropriate standard) for the distance between opposite flats of a Nut
  - need not be specified for a particular Nut

49. Washer:

A washer is a type of SteelStructuralFrameEntity that provides a description of a washer type, in terms of dimensions and other specifications and can form part of one or more BoltMechanisms.

Data associated with Washer:

- washerId
  - specifies a unique alpha-numeric identifier for a Washer

- standardWasher
  - specifies the category of Washer as
    - standard item
    - manufacturer's item
    - or non-standard item

- washerType
  - a text description of the shape of a Washer
  - need not be specified for a particular Washer

- boltDiameter
  - specifies a numerical value (in accordance with the specified units system and the appropriate standard) for the nominal diameter of a bolt that would fit a particular Washer

- insideDiameter
  - specifies a numerical value (in accordance with the specified units system and the appropriate standard) for the diameter of the hole in a Washer
  - need not be specified for a particular Washer

- externalDiameter
  - specifies a numerical value (in accordance with the specified units system and the appropriate standard) for the external diameter of a Washer

- washerGrade
  - specifies a short text description (in accordance with the appropriate standard) for the grade of the Washer
  - need not be specified for a particular Washer

- washerThickness
  - specifies a numerical value (in accordance with the specified units system) for the thickness of a Washer (i.e. the distance between the two faces)
  - need not be specified for a particular Washer

- washerMaterial
  - specifies a short text description of the material from which a Washer is manufactured
  - need not be specified for a particular Washer

- washerStrength
  - specifies the material strength of a Washer as
    - normal
    - high-strength
    - or special
  - need not be specified for a particular Washer

50. WeldSystem:

A weld system is a type of SJointSystem that provides a description of a welded joint system in terms of a specification and a *Trace*.

A Trace is a path in three-dimensional coordinate space. It comprises a sequence of straight line segments. These line segments need not be connected; they may make up a discontinuous Trace.

Data associated with WeldSystem:

- wsNumber
  - specifies a unique numerical identifier for a WeldSystem
  - need not be specified for a particular WeldSystem

- weldPath
  - specifies a trace followed by a WeldSystem
  - need not be specified for a particular WeldSystem
  - application assertion
    - each WeldSystem follows one Trace
    - each Trace specifies a path for zero, one or many WeldSystem objects

- wmSpec
  - specifies a weld mechanism associated with a WeldSystem
  - need not be specified for a particular WeldSystem
  - application assertion
    - each WeldSystem is specified by one WeldMechanism
    - each WeldMechanism provides a specification for zero, one or many WeldSystem objects

51. WeldMechanism:

A weld mechanism is a type of SteelStructuralFrameEntity that provides a set of characteristics for a weld.

Data associated with WeldMechanism:

- wmId
  - specifies a unique alpha-numeric identifier for a WeldMechanism

- electrodeType
  - specifies a short text description of the type of electrode used
  - need not be specified for a particular WeldMechanism

- standardWeld
  - specifies the category of weld as
    - standard weld
    - manufacturer's weld
    - or non-standard weld

- weldThroatThickness
  - specifies a numerical value (in accordance with the specified units system and the appropriate standard) for the thickness of the weld-throat
  - need not be specified for a particular WeldMechanism

- weldMaterial
  - specifies a short text description of the material used in a weld
  - need not be specified for a particular Weld_machanism

- weldType
  - specifies the type of weld as fillet or butt
  - need not be specified for a particular Weld_machanism

- weldDesignStrength
  - specifies a numerical value (in accordance with the specified units system and the appropriate standard) for the design strength of a weld
  - need not be specified for a particular WeldMechanism

- weldPenetration
  - specifies the penetration of the weld as
    - full or
    - partial
  - need not be specified for a particular WeldMechanism

APPENDIX B

LAYOUTS OF ENTITIES IDENTIFIED FOR FURTHER DEVELOPMENT:

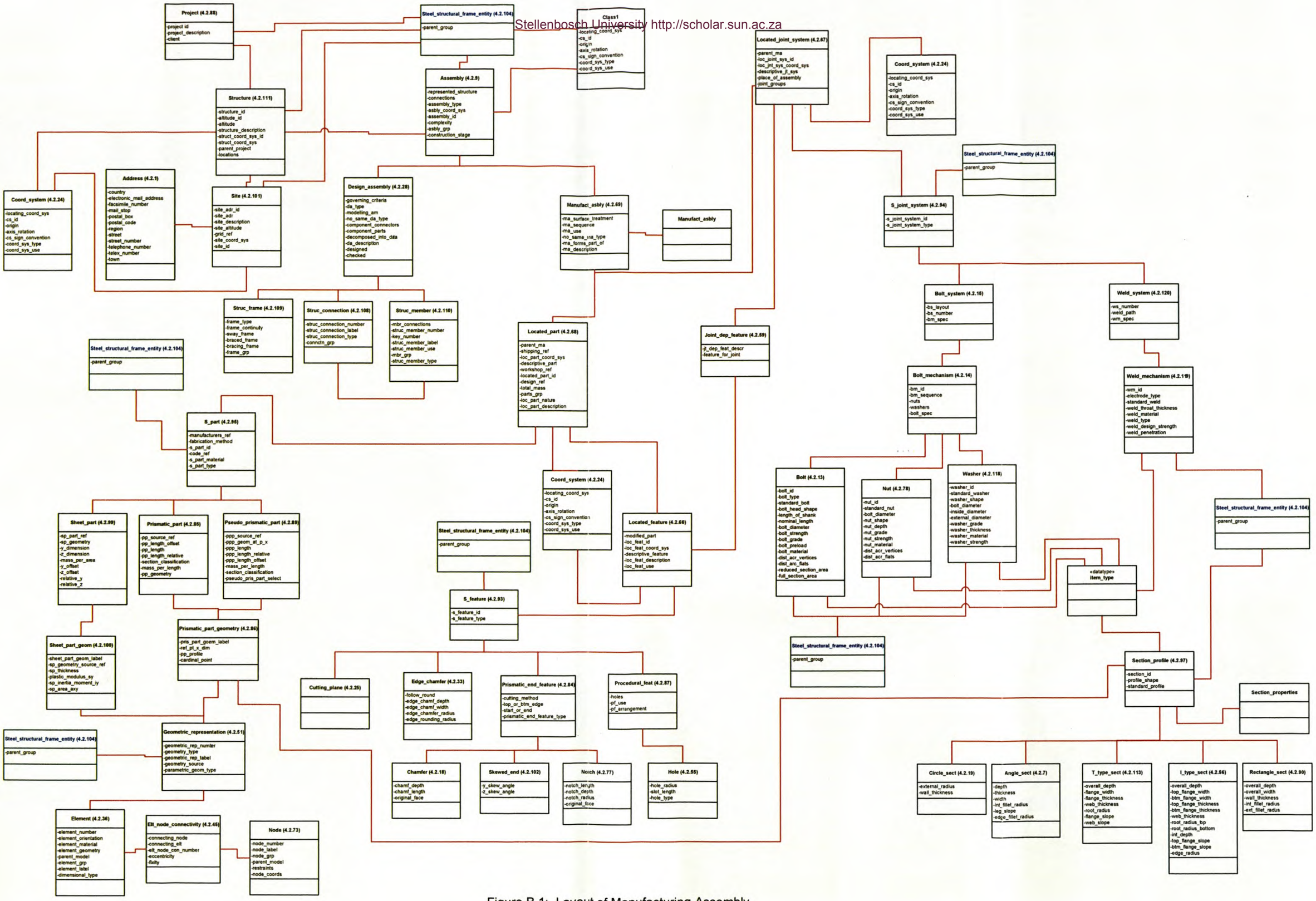


Figure B.1: Layout of Manufacturing Assembly

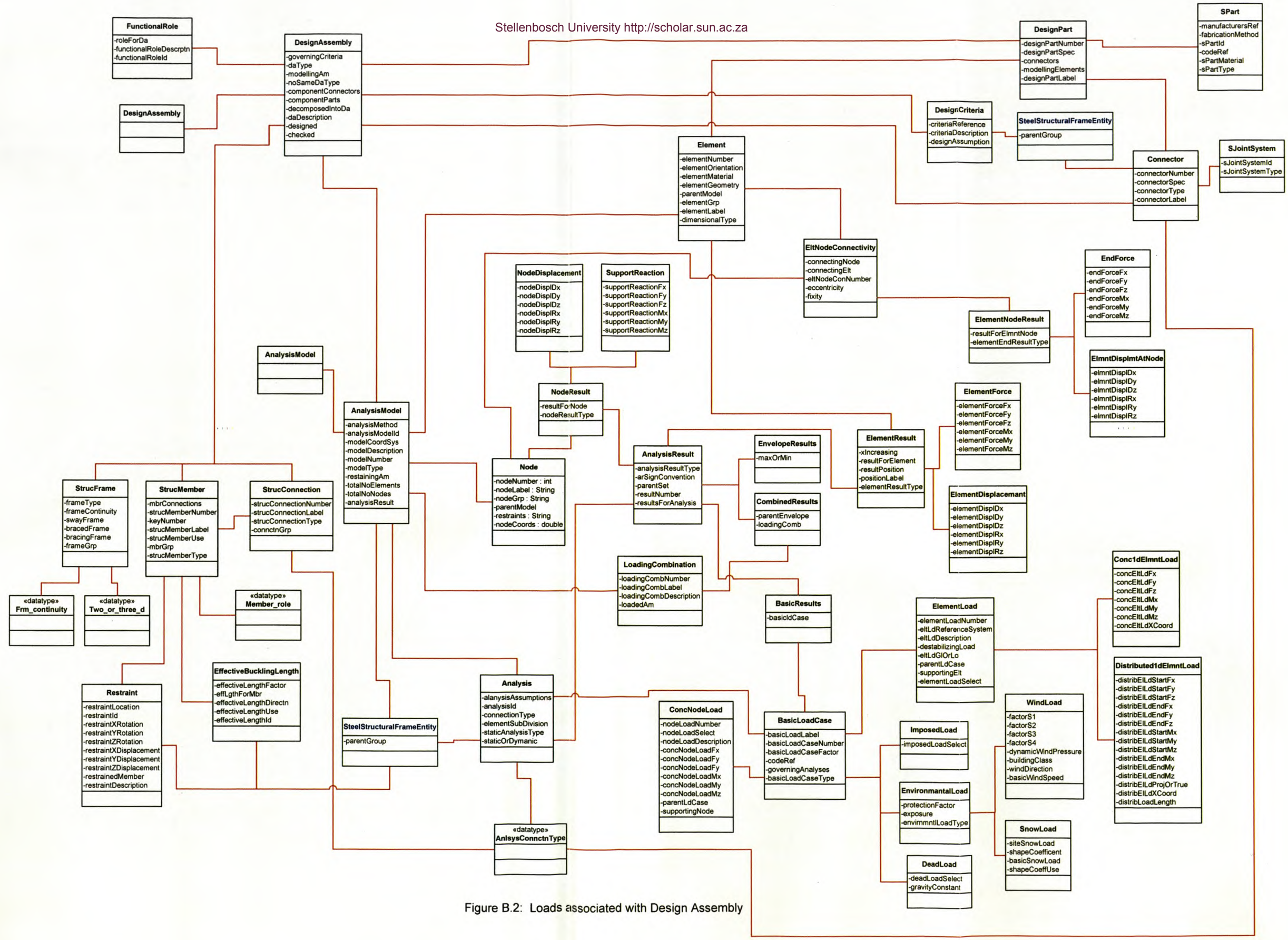


Figure B.2: Loads associated with Design Assembly