Modelling architectural design information by features: an approach to dynamic product modelling for application in architectural design

Citation for published version (APA):

DOI:
10.6100/IR524571

Document status and date:
Published: 01/01/1999

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
MODELLING ARCHITECTURAL DESIGN INFORMATION BY FEATURES

Jos van Leeuwen
Modelling Architectural Design
Information by Features

an approach to dynamic product modelling
for application in architectural design

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Eindhoven,
op gezag van de Rector Magnificus, prof.dr. M. Rem,
voor een commissie aangewezen door het
College voor Promoties in het openbaar te verdedigen
op dinsdag 29 juni 1999 om 16.00 uur

door

JOS P. VAN LEEUWEN

geboren te Hedel.
Acknowledgements

This thesis is the final report of a research project on information modelling for the support of architectural design. This research project was financed by the faculty of Architecture, Building, and Planning of Eindhoven University of Technology. The original research proposal was written by prof. Thomas Maver, PhD. The project was carried out with the support of many people. My first supervisor, prof.ir. Harry Wagter, has been a great support during the long process of learning, always transmuting from the devil's advocate into the perpetual optimist in harder times, and back again. Prof.dr. Robert Oxman, who is my second supervisor, has persistently supported me in structuring the project and the writing and by reminding me of the architectural context. I am much indebted to both.

Prof.dr. Harry Timmermans and dr.ir. Bauke de Vries have taken up the task of carefully reading the draft and participating in the committee. Their criticism is much valued. Bauke has been a devoted counsellor and loyal confederate. I am also grateful to prof.dr.ir. Erik Jansen, prof.dr.ir. Peter van der Veer, prof.dr.eng.arch. Herman Neuckermans, and ir. Godfried Augenbroe for being so kind to oppose during the defence of this thesis.

This project could not have been fulfilled, or even started, without the support of many who have been good friends and colleagues at various moments in the past years. My gratitude must go back to the time that Boudewijn de Bonth convinced me to rejoin him at university after my military service. To James Rutherford who was an inspiring example for me at Strathclyde University. And to my colleagues at A CAD Consult, who gave me the opportunity to develop a sound basis in CAAD. Particularly grateful I am to Aad Jongbloets, who in fact persuaded me to start this PhD study and became a friend whom I admire and respect.

I would like to thank all those at our faculty who have made it possible for me, directly or indirectly, to do this work. The group of Calibre / Building Information Technology, where this study was started, is now joined by the Design Methods Group, forming the new group of Design Systems. Here, I have found a most inspiring, enthusing, and creative environment of pleasant colleagues. Some of them in particular I would like to thank for their direct support in my work. Joran Jessurun has been great not only in helping to specify the syntax and the definitions in chapter 7, but also in developing the
second and third prototype. Henri Achten, besides his tireless interest and readiness to give an opinion, has importantly contributed by performing the case study reported in section 8.9. Marc Coomans has often surprised me by discovering yet another inconsistency or questionable solution in the FBM framework. Sjoerd, Marlyn, Jan, Rob, Jan Thijs, and all my other colleagues in the Design Systems group have been excellent in listening, debating, and assisting whenever I needed them.

Much indebted I am also to my friends and family, both Dutch and Portuguese, who have unconditionally supported and encouraged me. Their trust has brightened and strengthened me. Especially to my father I want to show my gratitude, he deserves all my respect. He has always been my greatest example, for his mastery, his persistence, his accurateness, his courage, but most of all for his personality.

Alexandra, you receive my deepest gratefulness and admiration, for always believing in me, for keeping me up and going, and especially for reminding me of life. Thank you for being so strong.
# Table of Contents

**Introduction**...1

Introduction to the field of research .................................................1

Introduction to the research project.................................................2

VR-DIS research programme: context of the project............................3

Structure of the thesis.................................................................4

**Part One: Architectural Information Modelling**..........................7

1. **Developments in Architectural Information Modelling**.................9
   1.1 What is architectural information? ......................................10
   1.2 Objectives of modelling architectural information ..................12
   1.3 Issues in modelling architectural information .......................14
   1.4 Approaches to architectural information modelling ..................16
      1.4.1 Digital documents..................................................16
      1.4.2 Geometric 3D models ..............................................18
      1.4.3 Integrating knowledge in modelling systems ....................20
      1.4.4 Object Oriented paradigm .........................................23
      1.4.5 Component based approach .......................................24
      1.4.6 Product modelling ................................................25
   1.5 Issues in modelling architectural design information ..............26
   1.6 Research questions ......................................................27

2. **The Product Modelling Approach**..........................................29
   2.1 Background and objectives of product modelling .....................30
      2.1.1 Alternatives for the integration of information systems .......30
      2.1.2 Background of product modelling ................................31
      2.1.3 Objectives of product modelling ................................32
   2.2 Concepts of product modelling ..........................................33
      2.2.1 Views, aspects, sub- and core-models ............................33
      2.2.2 Architectures for product modelling and data-exchange .......34
      2.2.3 Concepts of object orientation: objects and abstractions ....38
   2.3 Research and developments in Building & Construction ...............40
   2.4 Product modelling in the context of architectural design .........44
Table of Contents

3. Requirements for Modelling Architectural Design Information .............. 45
   3.1 The dynamic nature of design .................................. 46
   3.2 Extensibility ................................................. 49
   3.3 Flexibility .................................................. 50
   3.4 Implications for the design and development of design support systems .. 52
   3.5 Dynamic data models for design ................................ 53

Part Two: Feature-Based Architectural Information Modelling ............... 57

4. Origins of Feature-Based Modelling ..................................... 59
   4.1 Historical background ......................................... 60
   4.2 Definitions of the term Feature ................................ 60
   4.3 Feature taxonomies ............................................. 62
   4.4 The Feature modelling concept .................................. 64
   4.5 Approaches to Feature modelling ................................. 66
     4.5.1 Feature recognition ........................................ 66
     4.5.2 Design by Features ....................................... 67
     4.5.3 Feature conversion ........................................ 68
   4.6 Actual research topics .......................................... 69

5. Feature-Based Modelling in Architecture .................................... 73
   5.1 The essentials of the Feature concept for architecture ............... 74
   5.2 Features and similar concepts in architecture ........................ 75
   5.3 Problems for the analogy between the fields of application ............ 78
   5.4 Additional principles for architectural Feature-based modelling ....... 80
   5.5 Definition of the term Feature in the architectural context ........... 81
   5.6 Structure of a Feature model .................................... 83

6. An Inventory of Feature Types in Architecture ................................ 85
   6.1 Getting an insight in the extent and complexity of architectural information 86
   6.2 Classifications of parts and elements ................................ 87
   6.3 Product- and material documentation ................................ 91
   6.4 Architectural information revealed in product models .................... 93
   6.5 A case study on an inventory of architectural information ............. 98
   6.6 Conclusions from the inventory ................................... 108
   6.7 Categories of Feature types in architecture .......................... 109
     6.7.1 The case study continued ................................... 110
     6.7.2 A categorisation of Feature types ............................ 113

Modelling Architectural Design Information by Features
# Part Three: Framework for Feature-Based Architectural Information Modelling

7. A Framework for Modelling Architectural Features
   7.1 A framework for flexibility and extensibility
   7.2 Processes in Feature modelling
   7.3 The Feature-based information structure
   7.4 Generic versus Specific Feature Types
   7.5 A typology of relationships
   7.6 Twofold instantiation
   7.7 Classes of Feature Types
      7.7.1 The base class: FeatureType
      7.7.2 Subclasses of the class FeatureType
      7.7.3 Domains and defaults
      7.7.4 Feature Type Libraries
   7.8 Classes of Feature Instances
      7.8.1 The base class: Feature
      7.8.2 Subclasses of the class Feature
      7.8.3 Feature Models
   7.9 Instance level relationships
      7.9.1 Implications for the framework

8. Implementation of the FBM Framework in Architectural Design
   8.1 Strategies for Feature Type definition
      8.1.1 Identification of a concept
      8.1.2 Definition of the Feature Type
   8.2 Relationship issues
   8.3 Instantiation of Features
      8.3.1 Scenarios for Feature instantiation
   8.4 Dynamic manipulation of Feature Types and Feature models
   8.5 Standardisation and communication
   8.6 Interface-aspects: a metaphor for Features and Feature Types
   8.7 VR-DIS development: system architecture and implementation issues
   8.8 Prototyping the Feature framework in the Feature Management API
      8.8.1 Phases of the prototype
      8.8.2 Phase 1: Feasibility test
      8.8.3 Phase 2: Functional assessment
      8.8.4 Phase 3: Limited API development
      8.8.5 Phase 4: Full API development
Table of Contents

8.9  The application of Features in an architectural design case ............ 213
8.9.1 Example of Feature modelling in the case .................................. 215
8.9.2 Feature-based description of phase 0 ........................................ 218
8.9.3 Feature-based description of phase 1 ........................................ 222
8.9.4 A classification of design actions ........................................... 225
8.9.5 Conclusions from the case study ............................................ 226

Conclusions and Discussion .................................................. 229
Conclusions ........................................................................ 229
Discussion and future work .................................................. 234

Appendix A ........................................................................ 237
Appendix B ........................................................................ 243
Appendix C ........................................................................ 247

Literature References ......................................................... 255

Summary / Samenvatting ..................................................... 265

Index ............................................................................. 271
Subject index .................................................................. 271
Author index .................................................................. 275
Introduction

Introduction to the field of research

The discipline of Information and Communication Technology (ICT) has developed over the last few decades into a major commercial industry. The application of this new technology has challenged almost every professional discipline, in commerce, engineering, industry, service industries, medicine, education, humanities, et cetera. The successful development and implementation of tools and systems has become of the utmost important for these disciplines. It largely depends on the analysis of information needs and on the formal definition of information structures. However, after many years of intensive ICT development, the analysis and structuring of information still cannot be regarded as trivial tasks with predictable outcomes, not even for well-defined business-processes. These tasks become increasingly more important and crucial for success, also for less well-defined domains and processes. Especially in design and project-oriented processes, the information needs and the structure of information that is most suitable for a particular project are often known in full extent only when the design process has ended. The research project presented in this thesis concentrates on the problem how information should be defined and structured for design processes, keeping in mind the specific requirements posed by the way designers wish to handle information.
Introduction to the research project

This research project has started from the point of view that architecture is first of all a design discipline. Many efforts on information modelling relate to the support of production stages of business processes. Of course the architectural discipline has profited from this as well, not only by common administrative support, but in particular by more specialised application of computers in project planning and production processes. Production here means production of documents, e.g. drawings and bills of requirements, as well as manufacturing of building products. All of these forms of computer support can be well addressed in the way other business applications are developed. However, design requires a very different approach to computer support, since it involves creativity. Computer support must always be aimed at the enhancement of the primary tasks. Therefore, if this primary task is a creative process, computer support should stimulate creativity. At least it should not obstruct creativity.

In this research project the point of view is taken that design requires a dynamic way of dealing with information. Information models that are rigid, that cannot be manipulated in terms of the used definitions and structures of information, are not suitable for adequate support of this dynamic design process. At the start of this project, the need was acknowledged for an investigation of the requirements that must be fulfilled by information modelling tools for the support of creative design. On the basis of these requirements, the efforts on design support, both within the architectural discipline and in other disciplines, have been studied. Product Modelling (PM) at first seemed a very promising approach that already started to prove its value in a wide range of applications in the Building & Construction industry. However, it appeared not to meet the specific requirements for design support where creativity and a dynamic way of dealing with information is concerned.

From the mechanical engineering discipline, a modelling technology has emerged that, although developed contemporarily to product modelling, has quite a different approach in several respects. This technology, called Feature Based Modelling (FBM), allows a designer to produce models that appear less rigid and are based on less presumptions concerning the definition and structure of information describing the design.

In this thesis, the possibilities for application of the FBM technology in the problem area of supporting creativity in architectural design are explored. First, the requirements are surveyed that must be fulfilled for this kind of computer support. Next, the concepts from the FBM approach are reviewed and evaluated in the light of these requirements. An adaptation of the FBM approach to the particular conditions in architectural design appears necessary, this mainly concerns a broadening of the concepts to the wide range of information dealt with in this interdisciplinary field.

The intended contribution of this research project to the field of Information and Communication Technology in Architecture is twofold. Firstly, it aims to reflect in a
philosophical manner on what it means to support ‘design’ with computer tools; in particular what requirements must be fulfilled by information modelling approaches to truly support, rather than obstruct, creativity in design. Secondly, the project aims to develop an information modelling approach that meets these requirements and to define a logical framework that provides a basis for the development of innovative design systems that exhibit the advantages of this approach.

As a consequence of the stated objectives, the project is not very practical in nature with a high level of immediate applicability in the architectural design context. Yet, it is considered a necessary stage at the beginning of the development of innovative design tools to reconsider and possibly redefine the theoretical basis of the information technological approaches of such innovative tools. The outcome of this research project can be placed at a meta level of information modelling, which means that it does not in a direct sense define the working environment of an architectural designer, but rather the tools and conditions for the ICT developer that has the task to create this design environment. Prototyping, testing, and evaluation of the theory developed in this thesis thus take place mostly at the level of system-development tools rather than design tools. This has been an important factor in defining the scope of this thesis.

**VR-DIS research programme: context of the project**

At the faculty of Architecture, Building, and Planning at Eindhoven University of Technology, interdisciplinary design has become one of the major research issues. From its foundation, the various disciplines in the Building & Construction industry have been represented at the faculty, giving it a technological character. As design in this industry is becoming increasingly complex with more participants involved, each with increasing specialisation, individual responsibilities and often with different interests, the collaboration of these parties in architectural design becomes harder to realise. Not in the least this is due to the enormous amount of information that has to be dealt with by the members of such a design team, because of the increasing complexity of buildings, of building technology, and of the relations between the various disciplines.

One of the main research programmes at the faculty in this area is called VR-DIS [Achten et al. 1998]. This programme aims to develop design support subsystems for the various disciplines in Building & Construction, which are to be integrated in a single, though possibly distributed, design environment. The domain knowledge embodied by the individual subsystems can thus be combined in the integrated design system, under the responsibility and control of the participating designers and experts, in order to allow interdisciplinary design to take place.

The VR-DIS acronym has a dual meaning. From the point of view of application, it means Virtual Reality – Design Information System. This involves a virtual environment where interdisciplinary design can take place. It allows its users to creatively and intuitively build up a model of the design, incorporating all information that is considered relevant for the design into this model. Based on a comparison between
traditional computer aided design interfaces and the current possibilities offered by Virtual Reality technologies [de Vries and Achten 1998], the notion has risen that VR has a greater potential for successful development and application of such a design environment.

From a technical point of view, the acronym VR-DIS means Virtual Reality – Distributed Interactive Simulation. The term distributed indicates that the environment for interdisciplinary design does not require the participants to be physically present in the same room. Although communication with the other parties is very important for the functioning of a designer, there are many other factors equally or more important that will require another physical environment, for instance the building site. The term interactive marks the requirement of this design environment to allow participants to communicate with the system representing the design model as intuitively and direct as they communicate with each other. The last term, simulation, indicates that the design environment does not only represent a static model of the building being designed, but that also the behaviour of the building in its context, which includes its users, is simulated for purposes of evaluation of the design both by the designers and the future users of the building.

The research project presented in this thesis, although commenced several years before the initiation of the VR-DIS programme, has been embedded in the context of this programme. As the results from this research project started to emerge, it became clear that the VR-DIS related projects could benefit from its advantages. The requirement of supporting creative design, as set out for this project, harmonises with the requirements of the VR-DIS programme. The dynamic framework for design information modelling that has resulted from this project has many potentials for the development of the interdisciplinary design environment in VR-DIS.

Structure of the thesis

The thesis is organised into three parts. The first part deals with the objectives, various approaches, and requirements for modelling architectural information. The second part investigates the Feature Based Modelling (FBM) approach and studies the possibilities for adapting this approach to the problem area of architectural information modelling. The third part develops a technical framework for modelling architectural information with Features, and discusses the issues of implementing and applying this approach in architectural design systems.

Part One: Architectural Information Modelling

Chapter one starts with an overview of the developments in information modelling in the context of architectural design. The objectives for modelling architectural information are considered, followed by an evaluation of the various approaches that have emerged in research and practice of architectural design. This first chapter summarises the problems encountered with the diversity and complexity of architectural
information, which are elaborated in chapter three, especially in relation to the dynamic nature of design. This leads to the formulation of the research questions for this project at the end of chapter one.

The second chapter examines and evaluates product modelling, being one of the major approaches to modelling architectural information introduced in chapter one. It reviews the backgrounds and objectives of product modelling and its main concepts. Current research and developments in the area of product modelling are discussed and placed in the light of how to support architectural design.

Chapter three studies the dynamic nature of architectural design, and how this requires specific functionality from design systems that aim to support this dynamic character. The keywords in this functionality are extensibility and flexibility of the information model that is created to represent an architectural design. This chapter also includes a review of the work of other projects that address these issues.

**Part Two: Feature-Based Architectural Information Modelling**

Feature Based Modelling is a modelling technology that was originally developed in the field of mechanical engineering. The original theory, the concepts, and applications of FBM in their original context are reviewed in chapter four.

In chapter five, the FBM approach is evaluated with respect to the problem area of architectural information modelling. The concepts from FBM that seem relevant to the architectural context are distinguished and the problems for the analogy identified. Concepts developed in research in the architectural context that show similarities to the concept of Features are discussed. Chapter five leads to the statement of requirements for the application of the FBM approach in architecture and concludes with a definition of the term Feature in this architectural context.

The sixth chapter aims at getting an insight in the extent and complexity of architectural information. A selection of sources of architectural information are examined with the purpose of an inventory of typologies and relationships that are commonly used in architectural design. A case study on vertical space boundaries further narrows this study. From this case study, conclusions are drawn concerning the kinds of Features that can be expected to emerge in relation with vertical space boundaries. These conclusions are generalised into a proposed categorisation of architectural Feature types.

**Part Three: Framework for Feature-Based Architectural Information Modelling**

The framework for modelling architectural Features, introduced in chapter seven, defines a three layered schema for modelling architectural design information. The bottom layer represents the actual design data models, the middle layer contains the typologies for these models, and the top layer contains the meta definitions for both lower layers. This meta layer is developed in detail in this chapter, providing the formal definitions of classes of Feature Types, classes of Feature Instances, and their interrelationships. Also a graphical and textual notation is introduced for the representation of both Feature Types and Instances.
The last chapter discusses the implementation of the framework in architectural design. It describes how designers should be supported in defining their own Feature Types and what mechanisms should be incorporated in innovative design systems to fully implement the advantages of flexibility and extensibility of the framework. Chapter eight also includes a discussion of how the FBM approach can contribute to the development of standards for data exchange and how this approach compares to the major current efforts in this area. Since architectural Features involve both physical and abstract concepts of architectural design, the visual presentation of Features in a design system is an important issue of research. The considerations and actual results of a research project on this issue by Coomans are briefly presented.

Chapter eight then provides a description of how the FBM approach in this thesis has been adopted in the VR-DIS research programme as the basis for the information models underlying the design environment developed in this programme. It discusses the prototyping efforts that have been undertaken, partly within the context of the VR-DIS programme, on implementation of the FBM framework. The conclusion of this chapter presents an architectural design case study that uses one of these prototypes.

The thesis ends with the conclusions and a discussion of future work. The appendices contain a description of the notation techniques EXPRESS-G and WSN, and the collection of the definitions in the FBM framework as given in chapter 7.
As long as computer aided design tools cannot be adapted to the demands of the individual designer, they will be experienced as an obstruction for creativity.
Modelling Architectural Design Information by Features
Developments in Architectural Information Modelling

This first chapter reviews the research area of information modelling in the context of architecture. In an attempt to answer the question of what architectural information is, various aspects of this kind of information are discussed. Objectives of modelling architectural information are outlined and the general issues related with this are considered. Next, approaches to architectural information modelling are examined, varying from the initial introduction of digital documents and the integration of knowledge in modelling systems, to the area of product modelling. This is followed by a summary of the issues that are of importance when modelling architectural information for the purpose of supporting design tasks. While these issues are further elaborated in chapter three, they lead to the formulation of the research questions of this project at the end of this first chapter.
1.1 What is architectural information?

The first problem in answering this question is that in design-tasks the subject of the information involved, i.e. the resulting design, is not known in detail before the design has been completed. Also, the information defining the design-problem is often incomplete, ill structured, inconsistent, and even liable to change during the problem solving process of design. Although the domain of architecture has a rich terminology, the exact meaning of terms is rarely unambiguously defined, let alone formally structured. Yet formal structures of information are needed in order to support design by means of computers.

Subjects of information

Obviously, the subject of any piece of architectural information is always an architectural artefact or some kind of architectural concept. The domain of architectural information therefore is enormous and, in order to achieve some comprehension of this domain, a distinction between subjects of architectural information needs to be drawn. Such a distinction is not standardised, nor is there the intention to propose a standard at this point. However, in later sections of this book the standardisation of categories of architectural information will be a topic. The kind of distinctions considered below are not unique to architecture but are found in many disciplines of applied Information and Communication Technology (ICT).

First, different levels of abstraction can be distinguished in the architectural domain. Information is related to complete buildings, constituent parts of a building, components, spaces, materials, and so forth. A second distinction can be made between the physical parts of buildings and the non-physical concepts that play a role in architecture. Non-physical concepts include concepts of space, functions, relationships, costs, planning, etcetera. The different abstraction levels can be found in both types of concepts. A third distinction is between information that is related directly to the building and information that is related to the context of the building. The latter type includes information concerning for instance the building-site or its environment. A fourth important distinction is between project-bound information and information that is not related to a particular design case or building, but that is common or represents general knowledge. Examples of the latter kind are product information, material properties, physical principles, normative regulations and building-codes.

Views and aspects

Another common organisation of architectural information is by distinction of views on the architectural subject. Views represent the information relevant to a particular discipline or participant in the design and construction process, such as the principal, the architect, the constructor, the facility manager. A view does not necessarily cover the complete domain of a discipline. Within one discipline, different views may coexist. Tools developed for supporting the tasks of these disciplines are generally confined to
the information within the view of the particular discipline. Therefore these tools can be specialised and dedicated to this specific context, however, for the exchange of information between tools conventions and exchange procedures need to be applied.

The next categorisation of architectural information to be mentioned here is indicated by the term aspect. Aspects of an architectural subject are not confined to a particular discipline but are used to select similar types of information that may be relevant to several participants. Often, aspects are defined as properties of the architectural artefact or concept, for instance the cost property. Other aspects cannot be regarded so clearly as properties, for example planning aspects.

Views and aspects often coexist as mechanisms for structuring information. Section 2.2.3 shows that these mechanisms, along with the categorisation in levels of abstractions, form important tools in the definition of information models, particularly in the area of product modelling.

**Types of information**

The different kinds of information that have been mentioned so far can also be characterised by means of the typology of information. This typology is not necessarily related to or dependent of the media that are used to (re-)present information, although the typology has a strong influence on the selection and usage of the media. Likewise, the possibilities offered by traditional and new media have important influence on the selection of the typology of provided information. At a general level, the information typology includes textual, graphic, geometric, topologic, and geographic information, describing aspects such as shape, dimension, location, orientation, topological relationships, but also includes information in the form of pictures and images. Also audible information is getting more important with the technical advances of new media. Most other information is numeric, alphabetic, or alphanumeric information. Relationships between information can be regarded as a separate type of information, examples are the relationship between a beam and its support and the relationship between a wall and a space.

Procedural information is a type that is often called knowledge, since it represents methodical information on how to perform certain tasks, such as searching, analysing, interpreting, problem solving, and generating new information. Often information is presented in combinations of different types, a good example is the Internet where multimedia are used to present information of many different kinds.

**Quality and state of information**

Another perspective to consider when answering the question of what architectural information is, concerns the quality and state of information. The state of information involves several aspects, one of which is related to the stage in the design and building process. Design information in construction phases will have a state that differs from that of the same information in early phases of design. This state indicates whether information has taken a definitive form, or that it is still liable to change. Also, this state of information indicates if participants are entitled to modify information. A related
aspect is the source of information: normative regulations may have a different quality than information deriving from the principal, a manufacturer, or designer.

Another aspect related to the quality and state of information is the version of information. Especially in design processes, many variant solutions to a design problem will exist during the design process, probably even simultaneous. Management of these different versions of information can get extremely complicated, especially when multiple participants are involved in the design process.

1.2 Objectives of modelling architectural information

Many of the objectives for modelling information are common to all areas where Information Technology is applied. In general, it can be stated that the main objectives are to acquire less ambiguity in the way information is represented and stored, and to improve the speed and quality of retrieving, manipulating, and communicating information. Some objectives for modelling architectural information are discussed in more detail below.

**Enhanced representation and presentation**

The developments of new digital media and tools have allowed vast improvements in the way information can be represented (i.e. stored on some medium) and presented. The new media offer new ways of presenting information, visually, audible, and with tactile feedback, with increasingly realistic results. Access to information has become faster, while interaction with the representation has become almost commonplace. Also, digital media for storing information are more versatile in the way information can be presented and manipulated. Numeric information, for instance, is easily presented in both numeric and graphic forms like bar or pie-diagrams. Results of the physical evaluation of a construction can be presented in exact figures, or by means of colourful, scaled simulations in a three dimensional model. These new ways of presenting information require that information be formally defined and well structured.

**Formalised reasoning**

When using ‘weak’ representations of a design, such as pencil drawings and unstructured schemes and documents, these representations need constant interpretation when a designer is reasoning about the design. Without doubt, this has many advantages and may well stimulate creativity in design, moreover, for many designers it will be a most crucial part of the design process. However, once information about architectural artefacts and concepts has been formally defined and structured, reasoning about these artefacts and concepts can be done in a more formal manner as well. This means that designers can use formal methods for reasoning, for instance using knowledge from previous cases, style-rules, heuristics. This knowledge can be incorporated in design support systems, using computers to assist in the design process or to automate parts of the process.
**Improved communication**

In communication, a critical factor for success is that the meaning of information is transported from sender to receiver without losing the meaning of the message. This requires that both communicating parties understand well the domain in which information is being communicated. It also requires that information is provided in an unambiguous manner, meaning that after the transmission the information perceived matches the information that was originally sent. In order to meet these two requirements of achieving a better understanding of information and unambiguous transmission of information, formal definitions of information need to be organised in structured information models.

De Vries [1996] summarises the problems in communication in the building industry. Communicating information on paper is a time-consuming and error-prone process, errors occur when parties are copying information incorrectly, interpreting information differently, or overlooking available information. Although digital forms of documents eliminate the human copying errors, they do not change the need for interpretation, because information is still not explicitly defined but dependent of its context [De Vries 1996].

These observations lead to the conclusion that changing the medium of communicated documents from paper to some digital form is not an adequate solution to the communication problems. Instead, information needs to be expressed by means of proficient, explicit definitions.

**Better quality control**

Quality control is improved in different ways when using structured models of information. Firstly, structured information is easier to evaluate than unstructured information that requires laborious and error-prone interpretations prior to evaluation. Secondly, evaluation can take place on a more regular and directed basis, for instance by using formalised building-codes. It may be integrated in design support tools resulting in a direct feedback on design activities. Thirdly, supporting the design process with knowledge from systematic sources, like case-based systems, expert-systems, and generative tools such as rule-bases, shape grammars, and so on, may result in better quality control since the type of results produced by these tools are more predictable and therefore easier to control and evaluate.

**Enhanced design process**

With all the advantages described above, including the usage of computer support systems in design and direct evaluation of design decisions, the design process will benefit in the sense that more time will remain available to designers to spend on the actual creative tasks in design. Shorter cycles of analysis, synthesis, and evaluation result in more cycles, and variant-solutions, in the same time-span. Together with the improved quality control, this increases the probabilities of finding optimal solutions to the design-
problem. It may also help ensure that enough time is left for aesthetic aspects in construction projects that are often based on a commercial impetus.

**Easier management of design and construction project**

A last objective for modelling architectural information is to improve the management of information during the whole process of design and construction. Structuring the information and information flows and formally defining them in computer support systems helps in managing the process continuation, the consistency throughout the process, and can be a basis for many administrative and financial activities such as budgeting, progress administration, payments, et cetera.

### 1.3 Issues in modelling architectural information

As discussed in the previous section, information modelling can contribute significantly to the process of design and construction. The IT industry is to provide the adequate tools for this purpose. However, many issues are specific for the Building & Construction (B&C) industry compared to for instance administrative information technology applications. These issues require special attention in the development of the strategy and tools of IT implementation. This section summarises the most relevant issues, from the viewpoint of modelling architectural information.

**Ill-defined problems of design**

In many studies on the design process and possibilities for supporting design tasks, it is recognised that one of the major obstacles is that design-problems are often ill-defined problems [Simon 1973; Cross 1984]. This is partly due to the fact that architectural design problems are not pure rational problems, which can be solved in a scientific, analytical manner. Seldom is there an evident, indisputable solution to a design problem, mainly because the definition of the problem itself contains many ambiguities or even discrepancies. During the course of a design process, the original problem is often revised or supplemented in reaction to the feedback from the generated solutions. The final design suits the design problem not only because an optimal solution has been found, but also because the problem has been shaped while searching solutions. Therefore optimisation in architectural design is not a straightforward process. The fact that uniqueness is often a strong requirement in design also contributes heavily to this complication.

**Implicit information**

Much of the information in the B&C industry is implicit information. Implicit information exists only by way of some symbolic representation, for instance in drawings or texts. This can easily be demonstrated by a familiar example, such as a detail of a connection of a roof, a floor, and a wall. Figure 1-1 shows such a detail and with some affiliation to architecture it will be easy to recognise most of the materials implied by the drawing, including those that are not indicated in Dutch.
When formalising information into information models, implicit information can retain a similar status in the new representation media, that is implicit information remains implicit and is represented only by means of symbols or as a natural extension of explicit information. However, this requires that the knowledge to interpret these symbols and extensions of explicit information need to be formalised as well if automated tools, computer programs, are to be used to deal with information models. Explicit information is much easier interpreted in formal procedures. Therefore, it is probably more effective to also model implicit information explicitly, which means to include implicit information in the definition of information and to require that this information is explicitly provided during the design- and modelling process. An important drawback of this approach may be that the flexible interpretation of implicit information, which is often regarded a very important aspect of creativity in design, is easily lost when information always needs to be made explicit for support of the design task. Flexibility of information models is one of the main issues in the modelling approach developed in this research project, and will be discussed in detail in later sections.

**Complexity in design**

The complexity of designs in the B&C industry is comparable to the complexity found in industries such as aerospace and shipbuilding and is due to many reasons. For the design of buildings, the size of the product to be designed inevitably results in complexity. Multiple systems that compose a building, often show significant complexity themselves and require the maintenance of many relationships and dependencies within these systems and between systems. The different systems in a building are often the responsibility of different disciplines and participants, which increases complexity with complications of communication, planning, responsibility, ownership, et cetera. This is worsened by the uniqueness of projects in the B&C industry, with teams of participants being newly assembled for every project. Conventions between participants therefore are not easily formulated over longer periods.

Developments and researches on the automated support of design and construction are often criticised on the so-called aspect of data-explosion. This term refers to the phenomenon that when information involved in design is formally defined and structured, this would lead to an explosion of data and relationships between data. The author believes that this is generally a misconception, because the multitude of data
and relationships exists already before the formalisation in information models takes place, albeit that most of it has the form of implicit information. Therefore, the data-explosion is not caused by the implementation of information technologies, information technologies are the first attempts to actually tackle the problem of data-explosion that exists in design.

Integration of information

Much of the complexity of design in architecture is due to the many participants involved in the collaborative design and construction process. Communication of design information between participants forms one of the major problems. It brings along the problem of keeping consistent the total information used in different sub-processes of design. Integration of information and integration of tasks are important instruments in managing these problems.

Two kinds of integration are generally distinguished. Integration of information and tasks that belong to multiple participants is called horizontal integration. Since the composition of design- and construction-teams normally exists only for the term of a single project, horizontal integration will depend on the standardisation of information used by participants and standardisation of procedures constituting their tasks. Standardisation of information and application models is discussed in more detail in the next chapter. Vertical integration is the kind that integrates information and tasks in different (sequential) phases in the process of design and construction. Integration solutions will have to deal with the combination of horizontal and vertical integration problems.

The major problem in integration of tasks is that views on information in the domains to be integrated greatly differ between these domains, as do the formal definitions of information. A simple example to demonstrate this is to compare the view of an architect on a column in a space to the view of a structural engineer on the same column. The architect may see the column as a division or obstacle in the space, regarding it as an element between floor and ceiling, while the engineer may observe the column as a structural element that runs from foundation to roof. Clearly, the communication between both participants would be extremely confusing if one would not be able to understand the domain of the other. Integration of the two views requires that a modus is found for translating definitions from one view to another, or that a common view is defined that can be used by both parties.

1.4 Approaches to architectural information modelling

1.4.1 Digital documents

The first efforts on modelling architectural information have resulted in the replacement of traditional, paper documents by digital formats of these documents. These include documents like bills of requirements, design plans, building specifications, bills of quantities, budgets, schedules, regulations, building codes, manufacturer and product
documentations. Structure and contents of documents remained the same, just like their role in the design and construction process. Classifications of information in these documents, which have been defined for exchange purposes, are based on the structure of information in the conventional, paper documents. As a result, the demand and supply of information for the different actors in the process do still not correspond. Moreover, even today paper prints of documents form the basis for much of the communication and most legal contracts. One important factor in this conventional, document oriented approach is the fact that they still are the basis for national regulations that govern contracts for the B&C industry [Haas 1997].

The semantic value of documents has hardly improved with the move to digital formats. Interpretation of communicated information is as necessary as it was before, with the risks of errors, misunderstandings, and confusion. An example of this problem is shown by lacking integration of design plans and budgets. Information for the generation of budgets obviously needs to be derived mostly from design plans, but this relationship has not been made explicit in an integrated definition of information in both kinds of documents. Even existing classifications for both kinds of documents do not correspond, for they are based on different concepts and views on the building and on the construction process. For example, in the Netherlands many digital design plans are CAD drawings in which information is separated on the basis of the Dutch standard Elementenmethode '91 [SBK 1991] or NL-SfB\(^1\) classification for building elements. Table 1 in this classification distinguishes elements such as walls, doors and windows, columns, roofs, floors, and so on, in a fair detail. Tools for specification writing, such as the STABU\(^2\) methodology currently being used in the Netherlands, use a completely different categorisation of information which is based on construction activities. Retrieving information from the design plans for the purpose of specification writing still requires human analysis and interpretation of the plans, much too often even from the paper prints.

Some progress, however, has been made on the topic of structuring data in CAD models. A commonly used mechanism for structuring CAD data, is using so-called layering approaches incorporated in CAD systems. A layer is the metaphor used by CAD systems to resemble the usage of traditional transparencies in the production of 2D drawings. CAD models are built up from large collections of layers, which allows the user and reader of the drawing to select what information is to be shown on screen or

---

\(^1\) NL-SfB is the Dutch adapted version of the originally Swedish classification for building elements defined by the SfB, the Swedish committee for standardisation in the Building & Construction industry (Samarbetskommittén för Byggnadsfrågor).

\(^2\) STABU (STAndaardbestek Burger- en Utiliteitsbouw) is a foundation publishing the Dutch National Specification System for construction works and services, except civil engineering works. Although the system currently being used in practice is based on construction activities and tasks, STABU is taking part in the development of a new approach which has an object oriented basis [Woestenenk 1997].
Developments in Architectural Information Modelling

plotted drawing. Referencing techniques allow reusing CAD models in different contexts, which facilitates security and management issues. The value of the usage of layers in CAD models is obviously limited to a simple division method for otherwise implicitly represented design information (e.g., using different styles of lines, hatching, and symbols). For communication of CAD models, it is crucial that the naming of layers follows certain conventions. These layer names have been subject of standardisation at various levels, involving CAD user groups, large companies, and also the International Organisation for Standardisation (ISO 13567) [Björk at al. 1997].

1.4.2 Geometric 3D models

The development of Computer Aided Drafting systems has resulted in the availability of three-dimensional geometry modellers. Instead of sets of 2D drawings, designers are now able to create 3D geometric models of their designs, which can then be used to generate various forms of presentations of which 2D drawings (plans, sections, perspectives, details) are just one. The 3D geometry itself forms a more advanced presentation of the design, although a method for information separation is necessary in 3D models because of the quickly growing complexity. The most popular type of presentation resulting from 3D models is the rendered image used in e.g. stills (a single image), video animations (a fixed sequence of images), and real time simulations (Virtual Reality with direct interaction between model and observer).

Advantages and disadvantages of 3D geometric modellers

Apart from the enhanced possibilities for presentations mentioned above, the usage of 3D models, as compared to sets of 2D drawings, offers many other advantages. Relationships between information concerning e.g. plans and sections are intrinsically part of the 3D model. The task of designing in three dimensions is supported much better, offering more insight in the complexity of the three dimensions together and helping to maintain consistency in the design of details as well as on a larger scale. Consequences of modifications anywhere in the design become visible in a much more direct manner. As a result, errors during the preparation of the model and its interpretation can be significantly reduced.

Nonetheless, some new problems have emerged with the availability of 3D geometric modelling systems, while existing ones still have not been solved. The creation of 3D geometry is much more complicated and laborious than the preparation of 2D drawings. Although offering more insight in the geometry of a design, geometric models still do not allow semantics to be explicitly modelled. Moreover, the vocabulary used in geometry modellers, having a mathematical background, is very abstract and has no intrinsic relation to the domain of design. Elements such as columns and walls need to be modelled using terms like face, mesh, and vertex.

These problems have been addressed in several manners, such as the development of procedural and parametric geometry.Procedural programming and scripting languages have been and still are the main tools for developers of dedicated design-software. Integrated in CAD-systems, these languages can be used to automate much of
the tasks for generating complex geometry on the basis of design-decisions. This involves the capturing of design-knowledge into procedures of geometric modelling. Likewise, manipulation of geometry and maintenance of relationships can be programmed into procedures. Examples of procedural languages incorporated in CAD-systems are Autolisp (Autocad) and Toolkit (Visonael). The main limitations of this procedural programming approach are that the resulting model is still poor in semantics and that manipulations that have not been anticipated in these programs still remain very laborious. Despite the fact that some designers have taken up the challenge of creating their own dedicated tools, designers should not be assigned this new task of software development. Development of such tools requires a professional approach. An example of commercially available AEC specific customisations of the AutoCAD system are the design environments developed by the Dutch GB Software.

Parametric geometry is yet another approach to bringing the tools of geometric modelling closer to the domain of design. The model is created by means of geometric objects that are defined by parameters. These parameters represent variables deriving from the design-domain itself and the geometry is generated and manipulated on the basis of these parameters. This eliminates the problem for designers of having to translate design parameters into geometric entities and units. Although parametric modellers are widely used in for instance the car industry (e.g. Pro/Engineer, I-DEAS, Catia), it appears that, generally, in the AEC industry their potentials are still to be acknowledged. Although some of these systems also offer packages for the AEC market, most of this market is covered by less advanced (but also less costly) CAD products. An example of an AEC specific parametric model is Pro/Reflex by PTC.

Classifications in 3D

Classification of information remains a problem in any of the approaches of geometric modelling mentioned so far. Conventional classifications no longer suffice for several reasons. They are not prepared for the 3D geometry itself, not allowing the separation of information concerning for instance different floors in a building. Also, different levels of abstraction that can be modelled in 3D geometry, for instance using so-called logical zoom facilities, are not adequately supported by classifications that lack the specification of relationships between the different classes of information at different abstraction levels. The level of detail that can be obtained in 3D models often goes beyond the level

3 Autolisp and Autocad are trademarks of Autodesk (www.autodesk.com).
4 Visonael and Toolkit are trademarks of Visonael Corporation (www.visonael.com).
5 http://www.gbsoftware.nl.
6 Pro/Engineer is a trademark of Parametric Technology Corporation (www.ptc.com).
7 I-DEAS is a trademark of Structural Dynamics Research Corporation (www.sdrc.com).
8 Catia is a trademark of IBM (www.catia.ibm.com).
of detail provided in the layers of classification systems. This deficiency substantially limits the degree of semantics that can be added to 3D geometric models using these classification systems, and therefore restricts the communication between participants based on 3D models. Interpretation of the geometry by e.g. a quantity surveyor would still be essential.

Besides the development in design and modelling tools, the B&C industry has of course also developed new methods, techniques, products, and materials for construction. Existing classifications have been updated and extended over the years to the new modelling environments and to the developments in the industry. But new classifications that fully support the approach of 3D modelling and the modernisation of the B&C industry have still to be completed and implemented in practice. Especially the development of international standards for classifications proves to be problematic, because of the many differences between national standards, each based on national approaches and habits in representing building information and on national construction methods, products, and materials.

1.4.3 Integrating knowledge in modelling systems

Besides the development of procedural, geometric languages and parametric geometry, many other approaches to integrating knowledge into design support and information modelling systems have been developed. Many are still being developed and applied today.

Geometry linked databases

An early approach that has found wide commercial application since the late 1980's is linking databases to geometric modelling systems. This is a rather straightforward approach which, however, has proven a sound basis for many practical applications still in use today. The technology is simple, relating geometry in a CAD system to data in external databases. Most systems have implemented this by adding data from key-fields in a database to the geometric data, thus providing a one-way link from the geometry to the database that is to be managed by software running within the CAD system. Using standard access protocols for data-access, such as SQL and ODBC, the data management software provided with CAD systems can be fairly general.

Applications of this database linkage approach include GIS (Geographic Information Systems), Facility Management tools, and for instance production and manufacturing process support tools. One advantage of this approach is that the data is stored external to the CAD system and can be managed independently of the geometry by parties that have no interest in the geometric data. The consistency of such a system of storing and managing data in separate places is obviously more difficult to preserve.

This technique of linked databases has also been used to integrate CAD systems with other tasks of design management, such as quantity surveillance, requirement specification, budgeting, conformance checking, and so on.
Knowledge-based Systems

Knowledge-based systems, also called expert systems, are decision support systems that utilise formalised definitions of expert knowledge on a specific domain. These systems communicate with their users using the terminology of that specific domain. The knowledge can be represented in the system using various technologies, such as rules, constraints, or case descriptions [Netten 1997]. A number of approaches to the development of knowledge-based systems is briefly discussed.

Rule-based reasoning is a technique that uses mathematical logic in producing new states of a design from an existing state. Production rules that work on a given vocabulary, have been used in many different configurations of decision support systems. A typical form is the substitution rule used with a vocabulary of shapes. An example from the end of the eighties is the Topdown system that allowed the specification of syntactic rules and vocabulary of a parametric shape grammar [Mitchell et al. 1990]. One of the problems with this prototype system was that the design rules were relatively cumbersome to program. This issue has been addressed by van Leeuwen [1992] in a study on a visual programming environment for design rules of two dimensional parametric shapes. Shape grammars have also intensively been studied by, e.g., Stiny [1980], Flemming [1987], and Schmitt [1988] who describes the usage of shape grammars for generating designs in the style of, for instance, Mies van der Rohe.

Another issue involving shapes and rule-based reasoning is shape recognition. This technique attempts to recognise emerging shapes by application of rules that can interpret incomplete shape data or find close matches to known shapes. These techniques are used in, e.g., recognition of hand-made sketches [de Vries and Wagter 1990, 1991]. Many other examples of studies on shape grammars, shape recognition and emergent shapes are found in proceedings of the CAAD Futures conferences, e.g. [McCullough, Mitchell, and Purcell 1990] and [Flemming and van Wyk 1993]. A survey of spatial grammar technology is presented by [Krishnamurti and Stouffs 1993].

Another form of applying rules in design is by automating the process of code checking: evaluation of a design against the normative regulations. Recent work in this area has been published by, e.g., Dai and Oakes [1997] and by Stuurstraat [1997].

One strong potential of knowledge-based technologies is that computers can be made to 'learn from experience'. Machine learning is found in a variety of approaches, such as case-based reasoning, where solutions to problems are 'memorised' and can be used in future problem-solving tasks, and neural networks, where patterns of information can be recognised based on the degree of resemblance to concepts already known to the system. In general, case-based reasoning is based on making analogies between the current problem-solving situation and previously encountered situations, the cases [Maher and de Silva Garza 1996]. A case-based reasoning process involves two major tasks: recalling of relevant, previously known designs from the case-base, and adapting these design cases to fit the current design context. Building up a case-base not only involves identification of the cases and sources of information, it also requires...
Developments in Architectural Information Modelling

Structuring the information in a suitable representation format that depends largely on the goal for which the case-based reasoning system is to be used. Simoff and Maher [1998] present an ontology that categorises building design knowledge into the basic categories of activity (performed in a building) and space. This ontology includes a hierarchy of elements that further specify these categories, for example the elements time, performer, consumer, equipment, service, and constraints specify an activity. The ontology also includes relations between these elements and between activities and spaces. Using this ontology, designing a building involves designing the activity system, designing the relations between the activities and spaces, and designing the spatial system. On the basis of the activity/space ontology, a case-base is developed using structured multimedia documents in a web environment and so-called knowledge discovery techniques.

The approach followed by Oxman and Oxman [1993, 1994] involves the development of a cognitive model of case memory that is based on so-called design stories. Design stories are described in terms of design issues (the design problem statement), design concepts (the formulation of a solution in relation to an issue), and form (the design artefact that materialises the solution). They are formalised in a semantic network that is structured on the basis of these design stories, rather than complete cases: the case-memory is organised into smaller chunks of knowledge. The advantage is that this knowledge can be accessed cross-contextual, i.e. less dependent of the original context of the cases. Indexing the design stories further enhances the possibilities of knowledge retrieval. Two kinds of indices are used: search indices provide access to design stories that are relevant for a given design issue; browsing indices are based on design concepts, allowing linkage of different materialisation forms of the same concept. In [Oxman et al. 1997] the cognitive model of design issue, concept, and form is used as the basis for Internet representation of design knowledge. This allows the development of shared websites of design cases and introduces many additional advantages, such as graphical indexing of design case knowledge.

In [Gero and Nath 1997], a machine learning approach is proposed that not only acquires design knowledge, but also records the context in which this knowledge was learned, allowing the knowledge to be applied again when a similar situation in design occurs.

Knowledge-based systems are becoming more and more integrated in design environments, often in combination with other design specific problem areas of information technology. Rutherford describes a knowledge-based design decision support system that not only accommodates the access to design knowledge and information, but also facilitates the communication thereof, offering multiple interpretations of design data [Rutherford 1993]. This system uses a so-called blackboard architecture to allow communication between design participants, human or not, each responsible for domain specific tasks in the design process.

Defining conceptual design as an innovative reasoning process for configuration and parametric design, Netten proposes to support this reasoning process with an expert
system that aids in the performance of three consecutive tasks: prototype selection, concept selection, and concept modification [Netten 1997]. The reasoning process involves qualitative reasoning, allowing e.g. controlled search through cases and prototypes and ranking of cases and prototypes, followed by quantitative reasoning, used for e.g. numerical analysis and optimisation problems. The system proposed by Netten is a hybrid knowledge-based system in the sense that it combines various forms of knowledge representation and reasoning, using constraints, cases, and rules respectively for the support of the three tasks mentioned above.

Another research project combining knowledge-based reasoning with other fields in information technology is that of Lucardie [1994]. This project proposes an integration of knowledge-based technology, or artificial intelligence (AI), and database technology (DBT). Analysis of both fields at the symbol level, concentrating on the transfer of data and processes, reveals that AI involves inference processes using structures such as production rules, whereas DBT involves a model-based querying process of structures such as records. A similar analysis at the knowledge level ignores these distinctions at the symbol level and concentrates on the mathematical logic present in both AI and DBT approaches. Lucardie argues that following a symbol level strategy for the integration of both fields interferes with the process of modelling knowledge, since the relation between knowledge and knowledge representation is easily lost. The knowledge-based strategy in his research on the integration of both approaches defines knowledge as the functional object-types of a problem domain. Functional object-types can be defined either by means of probabilistic theory, prototypical theory, or functional classification. Lucardie reduces the problem of how to define functional object-types to that of finding functional equivalence of objects. This strategy uses a representation of functional object-types in a combination of decision tables and Prolog.

1.4.4 Object Oriented paradigm

The Object Oriented (OO) paradigm is a way of organising complexity. This organisation is based on the types of things - or object types - defining attributes of these object types, operations performed on or by these object types, rules based on these object types, and so on [Martin and Odell 1995]. It provides a way of organising knowledge conceptually, and therefore can be used as a basis for organising and interconnecting many other approaches to knowledge and information modelling, such as the ones mentioned above. An object can represent any concept that exists in the real world or in someone's mind. Booch [1994] defines an object as "simply a tangible entity which exhibits some well-defined behaviour".

Generally, three instrumental techniques are used to bring order in complexity: decomposition, abstraction, and hierarchy. Decomposition deals with the subdivision of a complex system into smaller parts with less complexity. Abstraction involves ignoring inessential details and concentrating on a generalised, idealised model of reality. Hierarchy of objects and object types defines their relationships and patterns of interaction. This involves recognising groups of object types with similar characteristics and behaviour.
thus allowing further abstraction and simplification of complexity. The OO paradigm is discussed in more detail in section 2.2.3.

1.4.5 Component based approach

The concepts of object orientation, as discussed in the previous paragraph, have been widely adopted. One approach that follows the OO paradigm in a rather straightforward manner is the component based approach, as described by Harfmann and Chen [1993]. A component in this approach is considered to be an individual, indivisible building element such as a nut, bolt, steel angle, an individual weld, a plywood panel, et cetera. Groups of components are not modelled as separate objects, in contrary to many other object oriented approaches, but are collected into units. The building itself is therefore not an object, but a very large collection of individual components. A 'reasoning mechanism' in this approach is based on three types of collections. 

- **Class collections** define rules and common information related to components or units with a certain similarity, e.g. the class of hinged doors. Individual components or units have access to the information in their class collection.
- **Assembly collections** contain general design process knowledge and presentational rules. This knowledge controls the generation of the component model and the way components are presented for specific purposes such as sections of plans. Finally, **functional collections** contain functional information related to assemblies of components. The physical relationships between components are stored within the component network, but the function of assemblies is represented by these functional collections. Examples are fire door assemblies and accessibility assisted passageways.

Modelling with the component based approach takes two directions. One is a bottom-up approach which specifies information on the detailed physical components as it emerges during design. The other is a top-down approach, generating information that describes the collections, for instance the functional collections, and gradually specifying the details of the components that form the embodiment in the building. The basic assumption for this approach is that the component model and resulting network must exist at the very beginning of the design process, regardless of how incomplete. The design task is finished only when the detailed component-level is entirely completed.

Potential drawbacks of this approach that have been recognised by Harfmann and Chen include problems with reasoning on the basis of incomplete data or for instance circular dependencies in reasoning. The example they give shows the mutual dependencies between the dimensions and structural performance of constructions. Also the representation of for instance monolithic building materials is recognised as a potential problem [Harfmann and Chen 1993].

---

9 In the component based approach by Harfmann and Chen, the building model is conceptually built up by individual objects. The implementation mentioned in the publication of 1993 is, however, based on relational databases.
Other problems with the component based approach, however, seem at least equally significant. The notion of hierarchy in the component based approach is present only by means of collections of components that represent the most detailed level. This bottom-up approach of modelling may easily become a problem in early stages of design when decisions on details are still to be taken and information on detailed components is not yet available.

Abstract information, i.e. information not describing physical objects, is not defined as entities in the component based data model, but as attributes related to collections of components physically present in the building. This also seems to be a somewhat limited approach for early phases of design where abstract concepts are the main subject of reasoning. The information that is available in early stages of design will rarely match, in terms of specifications and structure, the information defining the components in a fully detailed design. The fact that in early stages of design the information model depends on the availability of a network of physical components may well confine the view and creativity of designers, urging them to define their design in terms of physical solutions rather than in conceptual solutions.

Also at later stages, the definition of for instance functional information relies in this approach on direct relationships with physical components. It is the opinion of the author that this is a kind of decision on the process and method of design that should be left to individual designers. Whether a concept such as space should be modelled solely on the basis of its relationships with physically enclosing building elements is at least questionable.

1.4.6 Product modelling

Although parametric geometry modelling has already advanced the degree of semantics in geometric models, this was found to be still too limited for the purpose of advanced design support and for integration and communication purposes. Geometry cannot be used as a stable basis for the representation of product design information throughout the design and construction process [Luiten 1994]. Since the advent of object oriented techniques for analysis and development of information models and modelling environments, the concept of objects with semantic meaning incorporated in attributes has been applied at large, one example being the component based approach as discussed in the previous paragraph. Geometry in these ‘semantic object’ approaches has, at least conceptually, been demoted to the level of attributes of semantic objects. In practice, however, most systems follow a more hybrid approach to implementation, maintaining in parallel a geometric model and a semantic model, using for instance relational databases linked to the geometry.

The possibility to genuinely use semantics as the basis for information models has led to the conception of an ideal information infrastructure for design and productions processes. In this infrastructure, all information concerning a product, throughout its complete life-cycle, i.e. from its very conception via design, production, and usage until its demolition, is gathered and available for any interested party. The information model,
or set of information models, that forms the core of such an infrastructure is generally called a product model. Architectures have been developed for the technical and organisational implementation of product models, for example the General AEC Reference Model (GARM) [Gielingh 1988], the Bouw Informatie Model (Building Information Model) [van Merendonk and van Dissel 1989], and the Integrated Data Model (IDM) in the COMBINE project [Augenbroe 1995]. These, as well as the objectives and general concepts of product modelling, are discussed in detail in the next chapter. For now it suffices to mention that product models are being developed in many research environments, but few have been successfully implemented in detail and are actually operational in practice. Successful implementations have concentrated on selected stages and/or disciplines in the design and construction process, and therefore answer only in a restricted sense to the ideal as formulated above. The exchange of information in the ‘as designed’ stage between architect and engineers (structural, mechanical, HVAC) has mostly been the focus of attention.

### 1.5 Issues in modelling architectural design information

Many of the objectives for modelling architectural information, as summarised in section 1.2, refer to the enhancement of processes in architectural design. Modelling information during design tasks, and for the purpose of supporting design tasks, puts some specific requirements on the way information is defined and structured. It is generally expected that design support systems will develop into valuable tools for designers, also in the complex area of design in the B&C industry. However, this requires that these systems deal satisfactorily with the issues marked below.

- The problem-solving process in design.

Problem-solving in design often not only involves finding a proper solution to a

---

10 Architectural design: although a quite clear and unambiguous term in English, an equivalent term in Dutch is rather hard to find and has caused already much discussion (and headache) at Dutch universities. This is because two Dutch terms are related to the word ‘Architecture’. ‘Architectuur’ is most close by its sound and derivation. However, its meaning, in the opinion of many, only covers the part of architectural design that relates to finding the form of architectural artefacts. This is especially the case in journalistic usage of this term, where it also bears a sense of assessment or appreciation. ‘Bouw’ is a more general term which covers the totality of disciplines involved in the realisation of a building. The confusion in the usage of both terms is not much of a problem in the practice of the Building & Construction industry, but mainly becomes important when using them in relation to research and education. In fact, the very role of (future) architects is often buried in the discussion on the usage of either ‘architectuur’ or ‘bouwkunde’ (literally: science of building) as the appropriate indication for the field of research and education. It can therefore be concluded that the confusion is not due to a misinterpretation of the terms, but to an unresolved discussion on the changing role of architects in current and future design and construction processes. In this thesis, the term ‘architectural design’ refers to all those tasks in the development process of a building which involve activities of design, assuming that ‘design’ has the meaning of...
given problem, but also properly identifying the problem itself. Finding the problem in a design task often is part of the design task itself.

- Evolutionary development in design processes.
The process of formulating the problem and generating alternative solutions is an evolutionary process in which information is not static but invariably subject to change, both in content and structure.

- Stylistic evolution of designers.
Besides the evolution that takes place during a single design task, every designer also evolves during their career, meaning that their style of approaching a design problem, finding solutions, and consequently the way information is dealt with changes.

- The lacking will of designers to standardise.
Creativity and innovation in design requires that designers are not in any sense restricted by the tools available to them in the representation of their ideas. Standardised information models for design are therefore not willingly accepted by designers in general, because they feel that these models limit their freedom to choose how their ideas are conveyed, which often is regarded an important aspect of expressing creativity.

These issues are elaborated in chapter 3 on the requirements for modelling architectural design information.

1.6 Research questions

This chapter started with an observation of the various aspects of architectural information. It outlined the objectives for modelling architectural information and the various issues that it raises. Although product modelling offers many advantages and its potentials have been demonstrated by many research projects, it is not clear how it can deal with the issues summarised in the previous section that relate to modelling architectural design information. From these issues, the following research questions are formulated for the project that this thesis presents.

1. What should be the performance of (a) a model for architectural design information, (b) its structure, and (c) the modelling environment to support this structure?

2. Considering the required performance, how should an information model for the support of architectural design be structured?

This thesis aims to answer these research questions, by analysing the requirements imposed by architectural design on information models, and by developing an approach to modelling architectural design information that meets these requirements. First, however, the objectives, concepts, potentials, and limitations of the product modelling approach are studied in more detail in the next chapter.
The Product Modelling Approach

Product modelling is an approach to modelling the information related to a product in a manner that is aimed to cover the aspects of multiple disciplines involved with the product and from various life-cycle stages of the product. The concepts of this approach are general in that they can be applied in any kind of production process, including the Building & Construction industry. This chapter investigates the backgrounds and objectives of product modelling, its basic concepts and current research and developments in the B&C industry, in order to assess the suitability of product modelling as an approach for the support of architectural design.
2.1 Background and objectives of product modelling

The origin of the developments in the area of product modelling is found in the need to create models that are semantically rich, and in the need to structure the integration of computer applications used in product development. This section first reviews the alternatives for the integration of information systems of which product modelling is one, and then presents the background and objectives of this approach.

2.1.1 Alternatives for the integration of information systems

An inventory of organisational forms of communication between applications used in the design and construction processes in the B&C industry is given by Luiten [1994] in six distinct approaches. Although the items listed by Luiten seem to be of differing orders, some issues can be taken from the list that are important for a proper understanding of the state of art in integrating information systems.

- **Closed integrated systems**
  Communication is supported between parts of a suite of applications from a single vendor using a 'standard' specifically devised for usage within the system only. Communication outside the system is not supported.

- **Open integrated systems**
  The integrated systems consists of applications from different vendors that communicate using an agreed 'standard'. Communications to applications that are not part of the integrated system, i.e. applications that do not conform to the specific conventions, is not supported.

- **Communication with low semantic representations**
  Limiting the level of semantic content in the exchange information representation can help ease the development of standards for communication. This is done in both vendor-independent developments and industry standards such as the drawing exchange format, DXF, defined by Autodesk [1990]. Obviously, low levels of semantics require more human interpretation than richer kinds of data exchange.

- **Classification and coding**
  Classification and codes are the vehicles for document-based exchange procedures, as discussed in section 1.4.2. Classifications and standards for coding is the responsibility of national and international standardisation institutes, internationally this is the International Organisation for Standardisation (ISO) [ISO TC59 1993].

- **Product modelling**
  Product modelling is regarded as the approach with the best chances to succeed in communicating high levels of semantic content. Especially the object oriented basis for product models offers many enhancements to data-modelling and
exchange, allowing information definitions to take the form of objects that represent terminology and concepts from the targeted domain.

- **Knowledge-based technologies**

As opposed to product modelling approaches which concentrate on the definition and exchange of data, knowledge-based technologies focus on the formal representation of knowledge in order to support decision making. Knowledge-Based Systems (KBSs) involve the development of some form of reasoning mechanism (see also the discussion and examples in section 1.4.3). For KBSs to be communicating with (or even integrated in) data modelling systems, some form of interface between the data-definitions and reasoning mechanisms is required.

### 2.1.2 Background of product modelling

Product modelling has arisen from the need to create models of product data for purposes of enhanced communication and representation of information related to the product throughout its life-cycle. Formal definitions of product data have been used to describe the semantics of products, resulting in models that relate semantics to the shape representation of products. Traditionally, shape representations have been used as the basis for the model. Semantics was added to geometric models, either by means of 'attributes' added to geometric entities, or by means of external databases linked to geometric entities (see also section 1.4.3).

In today's practice of architectural drafting, commercial systems applying these techniques are still considered to be advanced systems. In research environments however (also those involved in architectural design research, although not forerunners), it has been realised in the late 80's that shape as the basis for models of semantic data is not suitable, mainly because of three reasons [Luiten 1994, p. 26].

- The shape of a product is not stable during the design process. This can easily be demonstrated by following the shape representation of a building element such as a wall from early stages of design where it may be represented as an element with one single thickness, to stages of detailed design in which the composition of the wall has been decided upon and the wall is represented as for instance a cavity wall with several layers each having a particular thickness and material.
- Often, information exists already before shape information is known. Examples of this kind of information are found in bills of requirements, including for instance spatial functions and relationships.
- Different participants in the design and construction process use different shape representations. A beam and column in the view of an architect, regarded as physical elements with certain dimensions in a space, may be very different from the way a structural engineer sees the beam and column, where the primary interest is in functions of the building's structure (see figure 2-1).
Object oriented (OO) methods and techniques have helped the development of approaches that address the above issues. One major advantage is that the terminology and concepts that are used in the domain of product design can be defined accurately in terms of object types in the OO model. Also the relationships between concepts are more explicitly defined as being a part of object types. Another improvement with OO techniques is that knowledge that previously needed to be represented separately from the data, for example behaviour, can be incorporated in object types, which enriches the data model with semantic meaning and results in more autonomous definitions. Geometry is no longer necessarily the basis for modelling semantic data, as was often the case with CAD-systems and linked databases. Instead, geometry can be defined as part of the objects model, taking the form of attributes of objects next to attributes with more semantic meaning.

2.1.3 Objectives of product modelling

The objectives for creating product models [Eastman 1993] can generally be summarised as follows:

- Provide a formal description of product data covering the information requirements of ‘all’ parties involved in the product design and production, during ‘all’ stages in the complete design and production process.
- Offer the definition of formalised semantics for storage of information and enhanced access to information by a multitude of approaches and computer-applications.
- Improve communication between design and production partners by means of higher level semantics of exchanged data and structured procedures for data exchange.
- Contribute to the development of standardisation of both data structures and exchange procedures.

With the development of OO techniques, the objectives in product modelling have come an important step closer within reach. The next section describes the concepts that are commonly employed within the OO approach to modelling product data.
2.2 Concepts of product modelling

2.2.1 Views, aspects, sub- and core-models

One of the major concepts in the development of product models is the notion of views. It has been widely understood that the many different participants involved in design and production of a single product each have a specific view on information related to the product. In architectural design, where the product is generally a building, this results in different views for participants such as the architect, structural engineer, HVAC consultants, and the like. Dealing with the same physical product, each of these participants has a particular way of describing the information representing the product (see the example on different geometric views in figure 2-1). Moreover, the set of information that is relevant for the different participants varies for each of them (e.g. a cost engineer might not be interested in geometry at all). These different views on building information lead to the definition of different information models for the particular requirements of each participant.

When communication is done by means of human interpreted documents, translation of information from one view to another is done during document interpretation. This requires that the involved individuals are able to read and understand the information in the documents of all those that take part in the communication. Computer assisted communication requires that the digital forms of exchanged documents can be read and understood by the software used on both sides of communication. This has been one of the main problems in software development and standardisation efforts in the area of CAD/CAM/CAE in the last two decades.

Product modelling efforts have incorporated the notion of views into the definition of information models and exchange methodology. In Building & Construction this has been subject of research by, amongst others, Eastman [1992], Rosenman [1993], van Nederveen and Tolman [1992], and Amor and Hosking [1993]. Different kinds of models can be distinguished in this field of research. Models that are defined for the purpose of a certain discipline, application, or view on a building are called view-models. Their scope and persistence is limited by the domain of the particular area of application. So the information concerning a single building will be defined in different ways in different view-models for different purposes of application. The fact that these view-models describe the same physical building is not modelled as part of the view-models. One way of defining this relation between view-models is by means of a

---

1 View-models are sometimes also called aspect-models, however, in this book the term aspect-model has been given another meaning which is explained in this section as well.

2 The relation between view-models can be defined also by means of mappings from one view to another. The usage of a core model is not necessary in this case. Some other approaches for organising the relationship between views and applications will be discussed in section 2.3.
so-called core model\(^3\). A core model could be regarded as a special kind of view-model, defining the information that is relevant to all or most of the involved participants in a way that is convenient, or at least agreeable, to all of them. This model should form the basis for the communication between the participants, meaning that it needs to contain definitions of the information that is to be exchanged between them. Various architectures of this kind of product model and several scenarios for the exchange of data using such a model exist and are discussed later in this section.

Figure 2-2 Concepts of product modelling
view-models, aspects, a sub-model, and a core-model.

Two more types of models need to be discussed here in the context of product models. As mentioned above, view-models represent the information regarding a building in different ways. However, often they may contain similar aspects of information, such as form, material characteristics, costs, or quantities. The information related to one single aspect, though distributed over different view-models, can be gathered in an aspect-model, for example a cost model containing all cost-aspects from each of the views of construction partners.

While view-models, core-models, and aspect-models include information concerning the whole building under consideration, models or sets of models may also be defined that regard only a part of the building. Such a model is called a sub-model. Figure 2-2 shows how the different kinds of models mentioned here represent different parts of the totality of information involved in a building's life-cycle. The next paragraph discusses some of the ways these different models can be interrelated and implemented into a system for data-exchange.

2.2.2 Architectures for product modelling and data-exchange

The relationships between the different models and types of models described above, can be organised into operational systems in various ways. Some alternative approaches for the exchange of data between these models and for the integration of data in these

\(^3\) Core-models are also called kernel models.
models have been identified by Hannus, Karstila, and Tarandi [1995]. These alternatives are described and discussed below.

**Inter-application mapping**

This is the traditional approach to data exchange, using translations, or mappings, from one data model to another. Each pair of communicating applications define and implement direct mappings between their data models.

*Advantages:* Semantic contents can be preserved to a high degree during data mapping.

*Disadvantages:* Multiple mappings are hard to realise and maintain, and are sensitive to changes in context (e.g., introduction of new applications).

*Status:* In practice this approach is still widely applied. The data formats of some leading applications have been promoted to 'de-facto standards' and therefore bring the model closer to the neutral model, as described below.

**Neutral model**

The neutral model takes up the role of application-independent intermediary that allows applications to communicate with each other associated by means of a single data mapping operation.

*Advantages:* Changes in related applications and additional applications do not necessarily lead to changes in the neutral model, therefore leaving the data exchange procedure intact.

*Disadvantages:* In practice, a neutral model is either clear and compact but inadequate for the desired level of semantic significance, or complex and voluminous with the result that it is hard to implement, maintain, validate, and not easily internationally standardised.

*Status:* A standardised, neutral model that applies to all of the Building & Construction industry has never been realised. For particular domains within B&C neutral models have been defined, often initialised by software user groups or based on de-facto industry-standards.
Application domain models

This approach involves the definition of neutral models, possibly standardised, that are valid only for a limited group of applications. Application domain models can be defined when the domain of the group of applications can be identified.

Advantages: Application domain models are neutral models of a smaller scope and are therefore easier to define, maintain, and standardise.

Disadvantages: Communication is restricted to applications within a particular domain. Data exchange between the domains is not provided for.

Status: Actually, this is the approach that is currently being implemented in the STEP standards, involving the definition of so-called Application Protocols (APs).

Common resources

The problem of isolated neutral models that cannot communicate is overcome in the STEP developments by using common resources as the basis for the different application specific neutral models. This means that applications can communicate across domains by means of their common basis defined in the resources.

Advantages: This approach avoids the tedious task of defining an industry-wide, standardised, neutral model that covers all exchange requirements between all possible applications in the industry. It allows application domains to communicate low level data representations.

Disadvantages: Higher level of data semantics can still not be easily communicated, since they will probably not be defined in the common resources.

Status: The efforts in STEP mainly focus on the development of the APs which are to be based on the so-called Integrated Resources (IRs). However, the common basis of the

---

4 STEP is the acronym of STandard for the Exchange of Product data, which is the informal name of the developments within ISO 10303: Industrial automation systems - Product data representation and exchange. The standardisation issues and the developments of STEP in particular are discussed in detail in section 2.3.
IRs is not sufficient for data exchange between different domains of applications, which is also indicated by the recent developments within STEP on standardised procedures for the translation of one application model to another, mapping technologies that are to be integrated in the EXPRESS data definition language. [ISO TC184 1994b; Amor and Hosking 1994; Liebich et al. 1995; Verhoef et al. 1995].

**Common core model**

This model represents the approach to data integration. In this approach, applications truly integrate their data models into a common core model, consisting of data definitions that are independent of any of the associated application domains. The data definitions in the common core are either independent of any of the application domains, or industry-wide accepted as being relevant for common use.

**Advantages:** This approach offers possibilities for true integration, allowing interaction between applications with a minimal loss of semantic content.

**Disadvantages:** Data defined outside the common core cannot be communicated without loss of semantics. Updating the definitions in the common core requires updating all associated applications.

**Status:** Model integration is found in practice at different levels of organisational implementation, for instance in closed and open integrated systems as discussed previously in this section.

**Mutually exclusive, common models**

Two applications sharing data may define a common model that is not of interest to any other application. This approach of exclusive integration combines the concept of data sharing with the peer-to-peer approach to inter-application mapping as described in model A in this section.

**Advantages:** Data sharing between two applications exclusively may involve high levels of semantic content and therefore considerably improve the communication between the two applications.

**Disadvantages:** The same problem exist as with the inter-application mapping approach: a costly process of defining and
maintaining the relationship between the two applications. Standardisation is most unlikely.

Status: This type of integration is more likely to be implemented in dedicated application-suites for domains that are highly integrated, for instance within a single enterprise or long term industrial partnership (closed integrated systems).

2.2.3 Concepts of object orientation: objects and abstractions

The general approach in product modelling for the definition and structuring of information representations in data models is to use the object orientation paradigm.

**Objects and object-types**

Information in object oriented methodologies is defined using data structures called objects. An *object* is a collection of data that represents a concept in the domain of the information model. The properties of an object are defined by the values of its attributes. These properties can be either static properties, describing the state of the object, or properties that describe the behaviour of the object, i.e. methods in object orientation.

A *concept* in this context can be something with a physical appearance, or an entirely abstract notion. The conceptual definition of objects, i.e. what type of information is defined in the object, is defined by an *object-type*, often called a *class* or *object-class*. Objects are instances of object-types and specify the values of the attributes defined in the object-types. For instance an object-type could be defined for the concept 'car'. The *car* object-type may have attributes such as colour, owner, fuel, and make. An object that is an instance of the object-type *car* will have values for these attributes, for instance green, john, diesel, and volvo.

**Abstraction mechanisms**

By definition, a *model* is an abstraction of reality. Abstraction means leaving out details of information that are irrelevant for the context in which we look at reality. In order to acquire a more structured representation of reality, several mechanisms are used to enhance the abstraction of reality with concepts that derive from the way we think about reality. These abstraction mechanisms generally define relationships between types of objects. Three such relationships are briefly explained here.

*Generalisation* is the abstraction of object-types which leads to the definition of a more general type of object by eliminating attributes from the object-type definition. The *car* object-type could be generalised into the type of object that we call *vehicle*. If we want the object-type *vehicle* to be defined general enough to include non-motorised vehicles as well, we would have to eliminate the attribute *fuel* from its definition. The relationship between the two object-types is an 'is a' relationship, so that any object of type *car* is an object of type *vehicle*. *Specialisation* is the reverse mechanism of generalisation, meaning that by adding attributes to an object-type's definition, a new, specialised object-type can be defined. For instance by adding an attribute *woman_or_man* to the definition of *vehicle* we could define a new object-
type for the concept of 'bicycle', with the result that, like cars, any object of type bicycle is also an object of type vehicle. The object-type vehicle is called the super-type (or parent) of the object-types car and bicycle, which in turn are the vehicle's sub-types (or children). The attributes that cars and bicycles have in common are defined in the vehicle object-type, and are derived from it by its sub-types. This derivation of properties is called inheritance, and applies to attributes describing an object's state as well as to attributes describing behaviour. Many OO methodologies allow an object-type to inherit attributes from more than one super-type. This is called multiple inheritance. Recent developments, however, show an increasing understanding that multiple inheritance will easily lead to over-complexity in software and data-models. For this reason, the newly defined OO programming-language Java, for instance, supports only single inheritance, which is generally not regarded as a shortcoming.

Another abstraction mechanism is called aggregation. Aggregations are used to indicate how parts compose into a whole. Its reversed mechanism is called decomposition. Both abstraction mechanisms help dividing the complexity of a given concept into manageable parts, allowing both a bottom-up (aggregation) and top-down (decomposition) approach to structure the information in the hierarchy of entities. In the above, the term attribute has already been used; defining the attributes of an object-type, where these attributes represent the parts of the object-type, is an act of decomposition. The creation of a new entity definition by collecting a number of entity-types and defining them as the 'part-of' attributes of a new concept, is an act of aggregation. Examples of this kind of abstractions can easily be given based on our daily environment. Wherever the phrase 'is part of' can be used, an aggregation can be defined. When 'has a' is applicable, a decomposition relationship can often be used: a book has pages; a page has two sides. However, care should be taken with the decision to call each 'has a' relationship a decomposition. The cover of the book has a colour, yet the colour is not part of the book. Usage of a decomposition relationships is only valid when the reverse, the aggregation relationship, can also be applied.

Finally, the relationships between object-types that are not generalisation- or aggregation-relationships, are called associations between object-types. This type of abstraction mechanism helps ordering information by allowing the specification of logical relationships between groups of information defined in object-types. The example of the book and the colour is such an association relationship, where the book can be associated with the colour and vice versa. An architectural example is the association between spaces and walls. Relationships between object-types are labelled by means of a name for the role of the relationship. This role name is often a verb: the walls bound the space. This implies a direction of the relationship, which depends on the verb chosen to name it. However, an association relationship can often be reversed: the space is bounded by the walls. This inverse relationship is sometimes modelled explicitly, but in many approaches implicitly available from the product data model.
The Product Modelling Approach

2.3 Research and developments in Building & Construction

Product modelling research-projects

In the last two decades of this century, many national and international research projects have been investigating and developing the possibilities of product modelling methodologies in the domain of the Building & Construction industry. These projects had varying backgrounds and objectives. Among the first of these developments were the Finnish RATAS project [Bjork 1989] and the General AEC Reference Model (GARM) [Gielingh 1988]. The GARM was one of the first attempts to define an international standard for product modelling. Its main contributions to the field were the distinction of functional unit and technical solution as individual elements in the product model, and the notion of stages of product data giving data a particular status, like 'as designed', 'as constructed', 'as used', etcetera.

In the first half of this decade, a number of EC funded projects have been developed on the basis of these early concepts. Each of these projects departed with a specific objective for modelling building information and with limited scope. The ATLAS project [Tolman and Poyer 1995] was developed within the problem context of large scale engineering. The COMBINE 1 and 2 projects [Dubois et al. 1995; Augenbroe 1995] started from the domain of energy and HVAC engineering, developing the Integrated Data Model. CIMSTEEL [Watson and Crowley 1994, 1995], as the name suggests, developed its Logical Product Model for structural steelwork frames. The COMBI project [Scherer 1995] mainly focused on information modelling for the support of structural engineering and foundation design.

The work of Eastman et al. [1995a] on the Engineering Data Model (EDM) is also to be classified as a product modelling development. Because this research, like the project described in this thesis, addresses more deeply the specific requirements of design support, it is discussed in chapter 3, after these requirements have been outlined. Also the BAS-CAAD project [Ekholm and Fridqvist 1997] develops a new approach to product modelling in a design context that is discussed at the end of chapter 3.

Efforts and results in standardisation

The international standardisation efforts in the area of product modelling have been concentrated in the last two decades in the developments in the ISO 10303 project, also known as STEP: STandard for the Exchange of Product data. Recently, another development has been started with a similar objective, but initiated by the CAD industry, under the name IFC: Industry Foundation Classes.

STEP develops its standard conceptual data models at two levels: Integrated Resources and Application Protocols. The Application Protocols (APs) are conceptual models containing data definitions that facilitate the modelling and exchange of data.
between a particular range of applications in various engineering processes. Such an AP is specific for the domain formed by the range of applications; it forms a common basis for exchange of data only between the applications that fall in this range.

In contrast to the APs, the Integrated Resources (IRs) are designed to be application independent. They are designed to serve as a common basis for the APs, in order to facilitate data exchange between the various ranges of application. Because the IRs are independent of a particular application domain, they are semantically poorer. All definitions of data that represent domain specific knowledge are included in the APs.

Apart from the development of the conceptual data definition models, the STEP developments have delivered three other categories of results. Firstly, the conceptual data models in STEP are formally specified by means of a data definition language, developed in STEP, called EXPRESS [ISO TC184 1994c]. This language has a graphical counterpart, called EXPRESS-G, which is also one of the notation techniques used in this thesis (see appendix A for the syntax of this graphical notation technique). Secondly, the actual data models based on the data definitions specified in EXPRESS must be stored in a neutral file format that can be read by the communicating computer applications. For this format another language is defined, called the STEP Physical File format (SPF) [ISO TC184 1994a]. Finally, the SDAI is the Standard Data Access Interface. It specifies an implementation method for a functional interface to data models that are defined using the EXPRESS language. These methods are specified in a manner that is independent of any programming language. Implementation of the methods is called language binding and is being developed in languages like C, C++, and Fortran.

The co-ordination between the definition of the APs for the various domains in a discipline, e.g. AEC, is currently not well developed. For exchange of data from one AP to another, applications must rely either on the usage of the Integrated Resources on which the APs are based, or on mapping mechanisms that translate data schemes from one AP to another. This has led to a situation where exchange of data between participants is very well advanced as long as they use the same application protocol, yet becomes a serious problem when the involved parties use computer applications based on differing APs. This issue is called the interoperability issue in product modelling.

One way to deal with this issue in STEP is to develop an AP that defines the common ground of the various APs in a particular discipline. One such AP currently under development is called the Building & Construction Core Model (BCCM) [ISO TC184 1996]. The BCCM is to be used as a reference by the other APs in the discipline of B&C, providing a common core for applications. This would have two important advantages: (1) common concepts in the discipline are indeed formalised as common

---

Application in this context means field where the model is applied, it is not to be confused with computer application or computer software, although in practice the APs are, of course, applied by means of computer systems.
data definitions, allowing the reuse of these definitions and therefore reducing development time of APs; (2) data exchange between applications based on differing APs can be made much simpler because of these common definitions, the parts of the model that is not common to APs must still be exchanged by means of mapping mechanism. If the development of APs with limited scope is considered a difficult process, the development of the core model appears to be a very toilsome task.

The Industry Foundation Classes (IFCs) are being developed by the Industry Alliance for Interoperability (IAI), a group of CAD vendors jointly designing domain specific data definitions, mainly based on the input from customers’ practices and third-party developers. As history has shown with for instance the DXF standard for CAD data exchange, this is a good basis for the development of a de facto standard. It is the software industry that takes the initiative and at the same time implements the standard in already wide-spread software. Also, the scope of the IFCs, being the Building & Construction industry, is much narrower than that of STEP, which comprises all manufacturing industries.

The main goal of the IFC development is to establish, in a cycle of releases, a working standard application programming interface (API) with the acceptability required for commercial feasibility of the project. The objective of immediate implementation potentially increases the development speed of the standard. Wide-scale implementation of the standards developed in STEP is not the responsibility of the STEP development and can only follow consensus of those involved in the time-consuming procedure of balloting the models. In contrast, the implementation of the IFCs runs much more parallel to their conceptual development and is also under responsibility of the IAI.

Another important factor for possible success of the IFCs is that this industry standard can build upon the knowledge developed in STEP and learn from the mistakes that have been made in this and the previous decade. Experts involved in the development of the STEP APs, such as the BCCM, are now also involved in the definition of the IFCs.

The IFC object model architecture provides different levels of data definition: the resources layer with least semantics at the basis, comparable to the level of the IRs in STEP, and the semantically rich domain models layer at the top, comparable to the APs in STEP. In addition, the IFC architecture defines two layers in between: the core layer, containing the concepts that are shared between the various domains, and the interoperability layer that facilitates the
adaptation of concepts in the domain models to the concepts in the core layer. The four layers form a strict hierarchy. The common basis for the various AEC domains in this architecture, formed by the core layer and the interoperability layer, promises to be semantically richer than the standards offered by STEP, at least for as long as an acceptable AEC-generic AP is not available.

**Discussions in the STEP community**

The crucial criterion for successful implementation of new ICT tools in any commercial discipline, is the adoption of the technologies by the practising market. The main threshold for the adoption in practice of any of the new approaches discussed in this and the previous chapter is that they require a change in design paradigm. Less knowledge-intensive tools, such as those for the support of document-generation, e.g. word-processors and administrative software, do not require such a dramatic change, and have been far more successful in being adopted completely by the market.

In the electronic forum discussion in the STEP community in 1997 on Information Sources for AEC Applications (ISFAA '97), Haas [1997] suggests that the ICT industry aiming at the market in the Building & Construction industry, in addition to long term developments, should follow an approach of less ambitious and more incremental implementation of new technologies in practice: short term improvements based on available technology. This approach would concentrate on the requirements formulated by practising experts. However, it is the experience of the author that new technologies require active introduction in practice. Demonstration of new concepts and methods is needed in order to stimulate their acceptance in practice and, more important, to initiate the discussion on new tools and to increase the awareness of their potential benefits in practice [van Leeuwen and van Zutphen 1994]. This is confirmed by others in the ISFAA '97 discussion [Martin 1997, Storer 1997]. It is the awareness of benefits that helped tools such as word-processors and for instance Internet tools to be accepted widely, rather than their similarity to existing tools, like typewriters and traditional means of communication [Haas 1997]. More important it is, however, to observe that the comparison of the requirements for IT in B&C with the requirements for word processors and like tools is not a very strong one. Word processors and Internet tools are not used to deal with semantics in information, which makes their task much easier. This kind of tool may be compared with the category of general purpose CAD software, rather than with the category of dedicated design support systems.
2.4 Product modelling in the context of architectural design

Returning, after this study of the product modelling approach, to the issues identified in section 1.5 on modelling architectural design information, it must be concluded that product modelling, in its current form, is hard to apply in processes of design. This can be deduced mainly from the following five problems of product modelling in relation with design.

1. The definition of information is prescribed in product model schemata, which requires that design-information is specified in a form that fits these definitions. Practice shows that this requirement is often a burden for designers in the case of architectural design.

2. Information definitions are provided in product modelling in the form of complex structures, for instance for the complete composition of building elements. This does not allow designers to give shape to the objects or concepts representing solutions that emerge from the problem-solving process of design.

3. Changing object definitions during design tasks is generally not allowed, only replacing objects with objects that may suit better the required performance is possible. The latter is an inefficient approach (because is requires more modelling activities than is necessary), and does not really represent the design activity (which is the changing of an object's characteristics, not a replacement).

4. Product models generally define a rigid hierarchy of architectural entities of information, which is prescribed for the design view. It is not necessarily true that the product model's hierarchy suits the view of designers, especially not during all possible stages of the design process.

5. The process of design itself, i.e. the contents of the design decisions, is not modelled in product models. An important disadvantage of this is that the reason of the decisions as they have been taken during design cannot be retraced in later stages of consulting the information models.

The general conclusion is that what is missing in the product modelling approach in terms of design support, is a design information model that (a) can be used to accurately represent the state of design during the design process and (b) can contain the semantics of the design paradigm and the meaning of design decisions.

These requirements for modelling architectural design information are elaborated in the next chapter.
Requirements for Modelling Architectural Design Information

Section 1.5 summarised the major issues in modelling architectural design information. Considering these issues the previous chapter led to the conclusion that product modelling in its current form is not a suitable approach to support architectural design. In order to define an approach that aims to adequately support design, the requirements posed by architectural design on information modelling must first be studied in greater detail. This study starts with the observation that design is very dynamic in nature, mainly because it is a problem solving process that involves creativity and evolution of designers. The various aspects of this dynamic nature are elaborated and conclusions are drawn concerning how information is dealt with during the process of design. This leads to the formulation of two main requirements for modelling architectural design information which are captured by the terms extensibility and flexibility. The implications of these requirements for the design and development of design support systems are discussed. The chapter ends with a review of research projects that address these same issues.
3.1 The dynamic nature of design

Design is a process of problem solving. In terms of information management, this means that design involves activities of searching information, analysing, manipulating, and structuring information, generating new information, and evaluating information. These activities do not form a sequential process, they take place in cycles. In the 60's and 70's, these cycles have traditionally been recognised in studies on the design process, describing phases of analysis, synthesis, and evaluation [Markus 1969; Maver 1970], and in the RIBA Architectural Practice and Management Handbook as phases of assimilation, general study, development, and communication [RIBA 1965]. Later, Lawson describes that cycles are inevitable in any map of the design process. Moreover, designers tend to switch in a rather ad hoc manner between the different phases in these cycles [Lawson 1990]. With a review of experimental research, Lawson indicates that there are no meaningful divisions to be found between analysis and synthesis. In other words, the activities of generating and manipulating information take place concurrently during design, with no predictable sequence.

The output of creativity in design results in generating information concerning the definition and specification of selected or generated solutions. It also involves combining new information with information that already exists in the design, finding new relations between known data and developing or discovering new structures in concepts and ideas that lead to design-solutions. Following the conclusion that activities of analysis and synthesis in design are concurrent activities, information is clearly not treated as static data, its content and structure are invariably subject to change. This means that an information model for support of design-tasks requires a flexibility that allows for (re-)definition and (re-)structuring of information: it requires evolution of the model during design.

Ramscar [1995] believes that product modelling approaches will not achieve a complete model that is necessary for an integration or exchange standard, because they result in models with a fixed view, which are unable to deal with unforeseen design data and cannot respond to the changing needs of their users. According to Eastman [1992], the weakness of product models with respect to design support is that they are insufficiently flexible to accommodate the diverse needs of designers and the changes in meaning during design. The notion of evolution and adaptation of models of design information is the basis for the development of the Engineering Data Model by Eastman et al. [Eastman 1995a, 1995b]. In the EDM research project it is argued that the schema for CAD data cannot be known beforehand, but is "defined incrementally as design proceeds" [Eastman 1991]. The structure of design data describing components in design depends upon decisions regarding the technology and functions associated with the design components. The work of Eastman et al. will be discussed in more detail in sections 3.5 and 5.2.
The Building & Construction Core Model in ISO 10303, or STEP [ISO TC184 1996] (see also section 2.3), perhaps not containing very large structures by itself, has the purpose to serve as a basis for Application Protocols, dedicated to a particular range of applications, that will consist of rigid models and are not designed to deal with unforeseen data-structures.

As a problem solving process, design involves the structuring of information concerning problems that are often characterised as ill-defined or even wicked [Cross 1984, p. 135-166], indicating, for example, a lack of explicit procedures for getting results, a lack of hard criteria for testing solutions, a large number of possible solutions to the problem, etc. Moreover, an ill-defined problem is often not known in full detail from the start of the design task and therefore its structure cannot be presumed. This is especially the case when many parts of the problem are interrelated while the relationships are not obvious. Chermayeff and Alexander [1963] stress that every problem has a structure of its own and that good design depends on recognising this structure and not easily presuming the design-problem to be similar to previously solved problems.

According to Simon [1973], a designer converts an ill-structured problem to a well-structured problem by reducing it to suitably organised sub-problems, and by adding constraints or sub-goals. These additional constraints and sub-goals are taken from the design-knowledge readily available to the designer and are used to improve the structure of the problem in order to find a suitable solution. Coyne et al. [1991] conclude that therefore the formulation of the design problem is itself dynamic, its representation being revised continually in accordance with the changing situation. The design problem is therefore always represented as a well-structured problem, but one that changes from moment to moment. “The formulation of the design problem can therefore be regarded as a problem-solving task in itself, one that is undergoing reformulation as the process continues. The operations by which states are changed are themselves in a state of change, as are goals, constraints, and evaluation criteria.” [Coyne et al. 1991]

Another aspect of the dynamic nature of design is that designers learn. They change their approach to solving design-problems, finding new techniques, new rules, new concepts. Mitchell refers to this as stylistic evolution, and stresses that CAD systems must provide for this essential component of creative design [Mitchell 1990]. A design system, according to Mitchell, is an “open, flexible, constantly-evolving knowledge-capture device rather than a static collection of familiar tools and dispensers of established wisdom”.

Design support systems must accommodate this stylistic evolution by allowing adaptation of the information models underlying these systems to the changing demands. A similar evolution can evidently be noticed in the Building & Construction industry as a whole, resulting in the constant development of new techniques, methods, products, and materials. These new agents in the field also require adaptation of information models for design support.

Modelling Architectural Design Information by Features
The following conclusions can be drawn from the above considerations of the dynamic nature of design.

- Design is a process of problem-solving, often concerning problems that are initially not well-structured.
- Information related to design problems and solutions is dealt with in different ways, related to the approach of solving the design problem. Design involves creativity through combination of these approaches.
  - Selection of an existing solution to a similar design problem involves matching information related to the problem and the existing solutions.
  - Creating a new solution to the design problem involves generating new information that defines the solution to the problem.
  - Combining existing information in order to find new relations or structures in concepts and ideas that lead to design solutions involves analysis, re-interpretation, and re-structuring of existing design information.
  - Altering the design-problem in order to find a suitable solution means analysis, re-interpretation, and re-structuring of the information related to the problem, possibly even adding, or dropping parts of the design problem.
- Activities in design do not take place in a predictable order, the information dealt with in design activities cannot be foreseen: the content and structure of required information or generated information cannot be presupposed.
- Individual designers, as well as the sector of design and the Building & Construction industry as a whole, are under constant development, with new knowledge, concepts, techniques, methods, products, materials, and styles emerging. Conceptual information models\(^1\) must evolve along with this development, in order to accurately represent the changing domain of design and B&C.

The above conclusions lead to the formulation of requirements of information models that are to support the dynamic nature of design. These requirements are denoted by the terms extensibility and flexibility, ensuring the possibility for an information model to

---

\(^1\) The terms information model and a conceptual information model relate to each other in a similar way as do the terms object and object-type. An object-type describes the common characteristics of the group of objects that are said to be of that particular object-type. Not the actual properties of objects are defined in an object-type, but the type of properties that characterise them. Objects are instances of object-types. In a similar sense does a conceptual information model describe the characteristics of the information models that are its instances: it describes the type of contents and the structure of these information models. For building models this means that, for example, a conceptual office model describes the type of information that is relevant to offices, while an instance of it, the model for Building X, describes the information related to Building X.
evolve along with the development of a design. Both requirements are discussed in detail in the next two sections.

3.2 Extensibility

For information to be represented in an information model it is necessary that the definition of information be formally specified in a conceptual information model. The formal definition of information includes the specification of the type of content of the information and the structure of the content. For instance, the formal definition of the information concerning chairs would specify the type of properties that characterise chairs: number of legs, colour, with or without armrests, headrest, et cetera.

In creative design, it is often the case that this typological information is not known in advance, that is, before information needs to be modelled for a particular case. In other words, concepts and notions in creative design cannot always be anticipated in generic conceptual information models²; they are often defined during design, by designers. Continuing the earlier example of chairs, the design of a chair with a tip-up seat may result in the definition of a sub-type of chairs that have additional properties.

Some situations are listed below in which the content and structure of information concerning a particular concept cannot be known prior to the moment this concept comes into actual usage:

- Information representing a specific design-style is defined by a designer and is in a constant state of change and extension during the learning process and career of the designer.
- The definition of style-rules or conventions on, e.g., dimensions and methods of construction for a particular building project can only be formally determined during the appropriate stages of the design and construction process of this particular building.
- The development of new techniques and methods of construction requires that representations for these new concepts be added to conceptual information models in use.
- The development of new products or materials to be used in construction requires formal definition of new kinds of characteristics that will be used to describe them. A notable example of the above situation is the development of industrial construction systems. The production of the individual components of such a prefabricated system requires a dedicated conceptual information model. Subsequently, the application of such a system in actual building design and

² This is why it is so hard to develop a standardised core model for AEC. Even for the stages in the B&CE discipline that supervene the design stages it appears to be extremely hard to define a core model that is sufficiently complete, of an adequate level of detail, and at the same time internationally acceptable.
construction requires information to be available that is strongly specific, in content and structure, for the particular building system. This includes information on the characteristics of the components and the system as a whole, the requirements posed on the design and construction with the system, and, for instance, assembly-procedures.

In most of these situations, concepts remain subject to change after they have been formally defined. For conceptual information models to evolve with these changing concepts, flexibility of the model is required, which is discussed in detail in the next section.

Another issue that pleads for the extensibility of conceptual information models is that the responsibility of the definition of information concerning a particular concept should rest with the designer of the concept, not with, e.g., engineers or vendors of modelling software.

### 3.3 Flexibility

One of the reasons why a conceptual information model should be flexible follows logically from the above discussed requirement of extensibility. Extension of a conceptual model requires flexibility: newly defined entities of information need to be embedded in the conceptual model. They will define relationships to existing entities, and conversely, existing entities will need to relate to new entities. This requires an adequate level of flexibility in the definition of information entities, allowing properties or attributes defining relationships to be modified or added.

Obviously, flexibility is also required when a conceptual information model is modified in other manners than by extension with new entities of information. This kind of restructuring the conceptual model could follow a change of insight in the role and function of parts of the model: a change in the meaning of concepts. The stylistic evolution of designers, argued by Mitchell [1990] to be an essential part of creative design, is an example of when such a modification of concepts occurs.

Another situation involving restructuring an information model, does not necessarily take place at a conceptual level, i.e., at the level of typological definitions. The type of flexibility that is required by this kind of modification of a model's structure is related to the flexibility in the way designers use their concepts. The relationships between concepts in a design, and indeed the meaning of concepts themselves, are rarely constant during the course of design; they are very likely to change from moment to moment as a designer changes point of view, has a new inspiration, or tries to find other possibilities in solving the design problem. This dynamic way of dealing with concepts and notions in design is regarded a crucial aspect of creativity in design [Coyne et al. 1991].

Adaptation of the conceptual model by changing the typological definition of a concept, does not necessarily result in the desired behaviour of the instance models. This is because a particular type of concept may be used in different situations within a single
model; its typological definition may be referred to in several different contexts. Modification of the typological definition of a concept according to the changing situation of one occurrence of the concept will easily lead to undesired effects in other situations where the same type of concept is used. Also it may not be a designer's intention to change the definition of the particular concept in response to non-typical application of it. This has to do with the different ways a designer may decide to apply a concept in a design. In a design model the following forms of a concept may appear:

- the concept in its typical form, i.e. exactly as it is typologically defined in the conceptual model;
- an application of the concept with omissions from its typological definition, i.e. some information defined for the concept is not relevant in the particular context in which the concept is applied;
- an application of the concept with additional information, i.e. information that has not been typologically defined but is relevant only to the particular context of the concept;
- an application of the concept with replacing information, i.e. part of the information that has been typologically defined for this type of concept is not applicable in the particular context, and replaced by other information;
- combinations of the three non-typical forms of a concept with additional, omitted, or replaced information.

For example the typological definition of a concept steel column may include the characteristics profile, length, position, upper-joint, and lower-joint. An application of this concept may provide the actual data, such as: HEA180, 6450 mm, at 350,200,2890 mm from grid-location C21e, bolts, welded.

Omission of information will normally occur in early design stages when information is simply not yet available, such as, in the above example, the profile of the column.

An example is an application of the above concept of a steel column, with the additional information that this particular column requires reinforcement between its flanges where the column connects to a beam that it supports.

An example of replacing information in a concept is a steel column of which the exact profile is not determined, but is instead confined by a maximum height of the profile, say 200 mm.

This form of concepts will actually be the most common form in design, since the context of design does not normally conform to the blueprint that has been defined in conceptual models.

These different ways of dealing with concepts in design, especially the latter three, determine extra requirements of flexibility in information models. They imply that
information describing a design is dealt with in a more dynamic manner than is defined in conceptual models. Besides the aforementioned flexibility required in conceptual information models, this dynamic way of dealing with concepts requires flexibility in the actual design information models, since the content and structure of the actual information deviates from the definitions in the conceptual model.

3.4 Implications for the design and development of design support systems

The requirements that ensue from the dynamic nature of design, above denoted by the keywords extensibility and flexibility, have significant implications for the design and development of information systems that are to support design activities. This section will introduce these implications as generic directives for the development of information modelling technology for design support. In chapter 5 and 7 they will form the basis for the information modelling approach and implementation framework respectively that are developed in this research project.

Extensibility of information models implies that a design support system must allow a designer to define content-type and structure of information at the conceptual level. This means that the data-definitions that the system works with are not limited to those predefined for the system, but can be extended by the user with definitions that better suit the particular circumstances in which the system is being applied. The system cannot be restricted to recognising a finite set of data-structures, but must also be able to deal with information that is not known by structure and content-type, prior to the moment that the user of the system defines it.

The task of defining the content-type and structure of information that is to be performed by the designer is not traditionally a task of an architectural designer. Therefore this task will require extensive support from the system, not only in the laborious activity of formally specifying content-type and structure of information, but also, for instance, in maintaining the consistency of information structures and in deciding when and what information should be formally defined. These are activities that require the kind of discipline, accuracy, consistency, and thoroughness that a designer, whose main interest is in architectural creativity, typically would not want to be bothered by. The technology developed for CASE tools\(^3\), for instance, has proven to be very useful in supporting tasks of information modelling. However, decision support in information modelling, assisting a designer in deciding what information should be modelled and how, is a much more complicated and undeveloped area that requires the incorporation of knowledge from the design-domain in the decision process.

\(^3\) CASE tools are Computer Aided Systems Engineering tools: computer programs used for the development of computer programs.
Modifications to information models and, moreover, to the definition of information models, require special attention since the consistency of both must be retained. This problem is generally addressed by a development denoted by the term \textit{versioning}, which concerns maintaining consistency of information models by keeping track of the different versions as they develop during a modelling process. This approach is applied in maintaining consistency of the contents of information models, yet if \textit{conceptual} information models are to be altered as well, the versioning problem is doubled. Though it is recognised that the versioning problem must be solved for dynamic modelling systems to work in practice, it is not considered a priority in this research project and therefore left out of the scope of this thesis.

Flexibility in dealing with information, in particular flexibility of design models (as opposed to flexibility of conceptual models), also implies special capabilities of design support systems. The system must support the addition of structure to an information model that is not defined at the conceptual level. This means that, somehow, the system must allow the user to add properties to entities of information that are not predefined, and still be able to manage the given information or even reason and infer with it. The representation of information in the system must be prepared for this kind of ad hoc information modelling and, for instance, browsing and searching facilities in the system should also take into account these unforeseeable data-structures. For example, when a designer adds an extra ‘cost’-property to a particular building element in the model, the system should recognise this extra cost and process it in its cost calculations. The knowledge to do so must, of course, be explicitly provided, either by the system-developers, or by the user, the designer.

### 3.5 Dynamic data models for design

The issues introduced in this chapter on how a model for design information should facilitate the specific requirements of design, are addressed also by a number of projects developed in academic contexts. Three of these projects are discussed in this section.

The objective of the \textit{B.AS-CAD} project developed at Lund University, Sweden, is to contribute to the development of CAD for early stages of the construction process \cite{EkholmFridqvist1996}. The approach followed to achieve these goals is by development of a unifying framework for conceptual modelling in the construction context, an independent representation of building information and information concerning the usage of the building. The project identifies three main classes of objects: system, property, and value state \cite{EkholmFridqvist1997}. A system object identifies a concrete thing in reality and has the attributes composition and internal structure to define the intrinsic properties of the system. The attributes external structure and environment define the extrinsic properties. Properties of a system are modelled by associating a property object with a system object.

Property objects are used to model the properties of a system. Properties can exist only once in a model, as in reality there can be only one conception of such a
property. This is to ensure compatibility between different objects in the model. For example, the property 'mass' can only exist in one definition, two different notions of such a property cannot coexist. Properties are further defined by a state function defining how the values of the property are determined, and a state space that determines the possible states of the associated systems with respect to the property.

The actual value of a property's state function for a specific system object is represented by state value objects. For example the property 'mass' of a system 'concrete' could take the state value '2300 kg/m$^3$'. The BAS-CAAD project is being prototyped in the object oriented environment of Smalltalk aimed at the modelling of both building design and user organisation in a lay-out process [Ekholm and Fridqvist 1997].

The EDM project started at the University of California in Los Angeles and is now hosted by the Georgia Institute of Technology in Atlanta. The project defines a data modelling language, the Engineering Data Model (EDM-2 in its second phase), that is "designed to capture the semantics of man-made products" [Eastman et al. 1995a]. This data modelling language can be used to define a database schema, to populate it with data, and to modify both data and schema over time. The EDM-2 project is the successor of the EDM project [Eastman et al. 1991]. The EDM project developed model evolution by allowing modification of a single integrated model. It supported the addition and subtraction of entities, attributes, and relations. New entity classes could be defined, while instances could be migrated from one class to another. The major problems encountered in this approach were that (a) modifications to a database schema rendered existing definitions for queries and application interfaces unusable, (b) it did not allow the designer much control over how the schema and instances were mapped as a result from the modifications, and (c) using a single integrated model was not useful in supporting collaboration in design [Eastman and Jeng 1998].

EDM-2 has another approach, which deals with the mapping between multiple application views and a central building model. In this approach, there is no single integrated model, but a central building model that is used for the communication between the various applications that surround it. The application views exchange data with the central model. Maintaining the consistency and integrity of the central model and the various application views during these sessions of data exchange and schema mapping has become one of the system's most important responsibilities [Eastman et al. 1997a, 1997b].

Clayton et al. [1999] have developed and implemented a product modelling approach that creates what they call a Virtual Product Model. This approach mainly concentrates on the flexibility of the product model and on supporting the assessment of a design. With a 3D CAD drawing of the design as a basis, the designer can use the SME program (Semantic Modeling Extension) to associate so-called features with the graphic entities in the drawing. This process is called interpretation and serves the execution of evaluation modules in the system. The features have a specific semantic meaning within the chosen
context of evaluation of the design, e.g. cost-evaluation. They are defined by the various evaluation modules that are attached to the SME system.

While the CAD drawing holds the form of the design, the features represent its functions. Several features may be associated with the same graphical entity, to represent different functions from different interpretation and evaluation contexts. After the drawing is been interactively interpreted by the designer in this manner, automated evaluation can take place which results in the determination of the predicted behaviour of the design.

The collection of form, function, and behaviour related with each single graphic entity in the drawing is gathered into so-called ‘virtual components’. The virtual components are not predefined as in other product modelling approaches, but are defined during design and evaluation, on the basis of the designer’s decisions and interpretations and on the basis of the outcome of evaluation processes. The components in the product model that thus emerge during design can be easily modified in correspondence with the ongoing design process.
Part Two

Feature-Based Architectural Information Modelling

"I wish I could sweep out your brain," Merlin once grumbled when Arthur was acting in a particularly muddled way.

"Why can't you sweep out my brain?" Arthur asked innocently.

"Because everyone and everything inside it is you." Merlin sighed. "You have turned into all those old, repetitive conflicts, and they will not disappear until you change."

The first step toward change is recognition.

From The way of the wizard, Deepak Chopra, 1995, p.68
Origins of Feature-Based Modelling

This chapter introduces the field of Feature-based modelling (FBM). This is an approach to product modelling that has been developed and applied mainly in areas of mechanical engineering. The concept of Feature-based modelling exhibits characteristics that make it potentially successful also in the area of modelling architectural design information, especially regarding the requirements that have been described in the previous chapter. A brief historical background of FBM is provided in this chapter, followed by a survey of how the concept of 'Feature' is defined and used in the main approaches that have been developed. This chapter is concluded by a review of the most relevant current research topics in the area of FBM in its original context of mechanical engineering. The potential value of the FBM concept and its relation to this research project are discussed in the next chapter.
4.1 Historical background

In disciplines of mechanical engineering and industrial design, the practice of using computers in design and production processes has given rise, in the early eighties, to the need for richer information models. Geometric models of a product did not suffice for advanced evaluation of designs, such as manufacturability evaluation, or for integration in, for instance, process planning tasks. Boundary representations (B-rep) provide low level geometric details of the surface of a shape by description of its collection of faces, edges, and vertices, and their relationships. Constructive Solid Geometry (CSG) is used to represent solids as algebraic expressions formed from a family of parametric primitives, using Boolean operations (union, intersection, and difference) and the operands' relative spatial positions. Although CSG can be considered a higher level of abstraction because shapes are represented in a purely symbolic form, neither B-rep nor CSG satisfy the designer's need because the level of abstraction, with terminology such as faces and vertices or blocks, spheres, and intersections, is still far from the designer's vocabulary [Hoffmann and Juan 1993].

For support of the actual design task itself geometric models were considered too low level because many aspects of the design task are not related to geometry at all, but to other characteristics of the product, such as the desired function and behaviour. Because low-level geometric models do not preserve the design intent of the designer, making changes to the design is very time-consuming; the information needed for propagation of changes through the model is missing [Shah and Mäntylä 1995].

Manufacturing of product-parts in mechanical engineering involves various machining techniques, such as twist-drilling, counter-sinking, milling, et cetera. For the planning of how a part can be manufactured using these techniques, the geometry of the part must be evaluated, in order to determine the appropriate kinds and order of techniques required. A method for doing this is called Group Technology (GT). With this method, a process planner assigns a code to a part to represent the subsequent types of machining that must be used to produce that part. This GT code is thus an abstract representation of the part: it represents a family of parts that can be produced using the same process. Historically, the assignment of GT codes to parts is a human task involving interpretation of the design of the part, but since this procedure can be represented by a decision tree, it is also supported using expert systems. For automated extraction of GT codes, interpretation and analysis of the part geometry is required in order to find higher level information about the part. Other tasks for which higher level information about a part is required are, for instance, NC programming, for numerical control of machining processes, and product assembly.

4.2 Definitions of the term Feature

The desire to extract higher level information from part geometry has resulted in the development of high-level information entities which are called Features. Since form is
the major aspect of interest in the disciplines where the Features technology was developed, a strong focus has been and still is on the development of Form Features. One way to get an understanding of what Features are in these disciplines, is to look at the definitions that have been given in literature for the term Feature.

The accents in the various definitions of the term typify the different views on a product for which the Features are devised. Some examples of such definitions follow:

- From a manufacturing point of view:
  "A feature is a geometric form or entity that is used in reasoning in one or more design or manufacturing activities." [Cunningham and Dixon 1988]
  "Features represent shapes and technological attributes associated with manufacturing operations and tools." [Shah 1991a]

- From a geometric modelling point of view:
  According to Bronsvoort and Jansen [1993], Faux [1986] defines a Feature as an area of interest on the surface of a part and Wilson and Pratt [1988] define a Feature as a region of interest in a part model.
  "A form feature may be defined as a recognisable geometric/topologic pattern and is defined in terms of either a set of faces and edges [B-rep] ... or as a set of primitive volumes and the operations needed to construct the part [CSG] ..." [Henderson and Chang 1988]

- Some definitions depart from the geometric point of view, but indicate the wide range of application of the technology:
  "A form feature is a physical element of a part that can be identified by some generic shape and attributes. A form feature has significance in design, analysis, manufacturing, or some other engineering domain." [Shah et al. 1990]
  "Features are generic shapes with which engineers associate certain attributes and knowledge useful in reasoning about the product." [Shah et al. 1993]

- When the accent of the usage of Features is on the support of the product design, the definition becomes much broader:
  "[A feature is] a set of information related to a part's description." [Shah and Rogers 1988b]
  "[A feature is] a functional shape aspect for design or manufacturing." [van Emmerik 1990, p.92]
  "Features are elements used in generating, analysing, or evaluating designs." [Shah 1991a]

In [Shah and Mäntylä 1995, p.8] the following definition is used:
  "[Features are] generic shapes or characteristics of a product with which engineers can associate certain attributes and knowledge useful for reasoning about that product. Features encapsulate the engineering significance or portions of the product geometry
and, as such, are applicable in product design, product definition, and reasoning about the product in a variety of applications such as manufacturing planning."

[ibid. p 10] "At an abstract level, we can view features as modeling entities that allow commonly used shapes to be characterized and associated with a set of attributes relevant to an application. In this sense, features may be thought of as information clusters or 'chunks', [...] packets of related facts and characteristics of a part."

In relation to product and process models, at generic and specific levels subsequently, Features address the following functionality:

- At the level of generic product knowledge, Feature types or classes are used to model recurring characteristics of products and as a repository of reusable product knowledge that may be related to a particular shape or geometric pattern.
- At the level of product models, Features build up the model of a specific product, providing a more natural basis of interaction with the designer than mere geometric models. Generic knowledge may be accessed through the Features recorded in the resulting product model.
- At the level of generic process models, manufacturing knowledge can be associated with Features and accessed to determine the producibility of a designed object or for planning its actual manufacture.
- At the level of factory models, that contain instances of particular processes, the capabilities of factories can be recorded on the basis of processes, which in turn are related to Features. Features provide a natural separation between design and manufacturing domains [ibid. p.8].

### 4.3 Feature taxonomies

Perhaps as many taxonomies of Features have been proposed as have been definitions of the term Feature. Based on various criteria, categories of Feature classes have been distinguished. One such criterion is the application or stage in the product's life-cycle for which Feature classes are defined. This generally leads to categories like:

- design Features, representing the terminology of the designer;
- evaluation Features, for evaluation of the quality of the design;
- manufacturing Features, related to the techniques and tools of machining the part;
- assembly Features, describing the characteristics of the parts that allow them to be assembled into a product;
- planning Features, for planning processes such as manufacture and assembly.

In general, the distinction between design Features and manufacturing Features is emphasised, as in [Meeran and Pratt 1993; van Emmerik 1990; Chamberlain et al. 1993].
Another categorisation is based on the function of Features, leading to categories of, e.g., form Features (nominal geometry), material Features (composition and condition of material), precision Features (tolerances and surface finish), technological Features (related to performance and operation of a part) [Shah and Rogers 1988a], and categories of pattern Features, connection Features, property Features, application Features [van Emmerik 1990, p.92]. These categories coexist with the previous ones. Also, similar Features will appear in multiple categories, but with different meanings. For instance, a cylindrical hole Feature would appear as a functional form Feature in design, as an analysis Feature in finite analysis, as a process planning Feature for boring operations, and as an assembly Feature for connecting parts [Bronsvoort and Jansen 1993].

Other distinctions are made between elementary Features and compound Features. Elementary Features, or atomic Features, are simple and cannot be decomposed, whereas compound Features are composed of several other Features [Bronsvoort and Jansen 1993]. An example of a compound Feature is a stepped hole, which is formed by two concentrical holes with different diameter and depth. Shah and Mäntylä [1995] also discuss compound Features. Together with pattern Features, which are recurrences of Features, they form a category called composite Features. The usefulness of composite Features is related to the manipulation of Features at multiple levels. By means of a composite Feature, a group of related Features can be manipulated as a single unit. Composite Features must therefore have their own representation in a Feature model [Shah and Mäntylä 1995, p.107].

Bronsvoort and Jansen [1993] further distinguish between implicit and explicit Features, both within the category of geometric Features. Implicit Features, or unevaluated or procedural Features, are Features that are described by parameters, but that are not evaluated into an explicit geometric description. An example is a Feature representing a screw by parameters for holding the type and size of the screw. Its exact shape, especially of the screw thread, is not represented, it is merely implied. Explicit Features, on the other hand, are evaluated and their shape is explicitly described by a geometric model. The most important advantage of implicit Features is that less detail occurs in a model, since only the parameters are stored, not the complete details of the resulting geometry.

A conclusion generally drawn from attempts to classify Features, is that such classifications are never finite and always application dependent.
4.4 The Feature modelling concept

The main idea of Feature modelling is that Features are used to describe the properties and function of a part of a product. The design process in mechanical engineering is normally seen as a top-down process of design involving phases such as:

- functional design;
- conceptual design;
- system-level design (overall layout);
- detail design (module- and component-level).

These phases are followed by other phases of the product’s life-cycle:

- manufacturing process planning;
- modular manufacturing;
- final assembly of the product;
- installation, maintenance, and upgrading;
- disassembly and reuse/recycling of the product [Shah and Mäntylä 1995, p.92].

The major role of Features in mechanical engineering is in the italicised phases above, which are those phases where the most detailed description of the product is important. Products are defined as assemblies of parts that are indivisible, the smallest manufactured elements of the products. These parts are modelled using the Feature modelling technique; Features are the constituents of parts and parts are the constituents of product assemblies [ibid., p.98] (see figure 4-2).

Features are defined by their properties [ibid. p.100-106], which can be either intrinsic or extrinsic. Intrinsic properties are properties that affect only the Feature itself, whereas extrinsic properties affect also other Features. Typical intrinsic properties are a Feature’s name, its geometric shape, and its unconstrained dimensions and tolerances, e.g. for the diameter of a cylindrical hole. Extrinsic properties depend on the properties of other Features. Typical examples of extrinsic Features are constraints on Feature size, location, and orientation, such as the position of a hole that is concentric with another hole. Extrinsic tolerances are those that involve properties of other Features.

Another distinction in kinds of properties is made between derived and non-derived properties. The values of such properties are not determined by the user, but derived from other Feature properties according to derivation rules. These derived properties may be needed for the creation process of a Feature, or

![Figure 4-2 Product hierarchy in mechanical engineering, after Shah (1991b).](image-url)
for evaluation or validation of the Feature. Derived properties are intrinsic when they depend solely on intrinsic properties and extrinsic when properties of other Features play a role.

An important aspect in Feature modelling is the usage of constraints, for instance for relative positioning of geometric Features in relation with other Features. One way of establishing positioning constraints is by using reference Features for the definition of the co-ordinate system of a Feature. This method of positioning a Feature has the advantage that modifications of Features are automatically propagated to related Features.

Although research is being conducted on the application of Features for assembly, the major role of Feature models is to describe the individual parts of a product. The organisation of a product's model can thus be viewed as in figure 4-2. Features are used only to describe the lowest level of detail in the complete product model. A constraint between two Features that belong to different parts is called an assembly Feature. The advantages of treating these associations between Features of different parts in an assembly as Features by themselves are that:

- multiple constraints between two Features can be grouped into a single unit,
- the constraints can be expressed in the same terminology of Features;
- changes to the Feature model of one part can now be propagated to other parts of the assembly [Shah and Mäntylä 1995, p.179].

Van Holland and Bronsvoort [1996] and van Holland [1997] describe a uniform modelling environment for modelling single parts and assemblies. The single parts are described by Feature models that are built up from Features with constraints as the relationships between Features, whereas the assemblies are described by combinations of components that are related by connection Features.

![Figure 4-3](image)

Figure 4-3 Combined data structure for modelling single parts and assemblies, interpretation of the work by van Holland [1997].
However, other than these inter-part Feature-relationships, the functional, more abstract characteristics of the assembly as a whole are not modelled using the Feature technology but with the general product modelling techniques as described in chapter 2.

4.5 Approaches to Feature modelling

Geometry and Feature data are, at least conceptually, separated in the approaches of Feature based modelling. Two categories of approaches can be distinguished for the creation of Feature models. This distinction is based on the kind of relationship that exists in these approaches between the geometry and the Feature data. The basis for this distinction is whether the Features are extracted from the geometry, or whether the geometry is created from the Features; the first method is called Feature recognition, the latter design by Features. Shah and Mantyla [1995, chapter 4] provide an authoritative review of these different approaches. This section's summary of these approaches is mainly based on this source.

4.5.1 Feature recognition

Historically, Feature recognition was developed as a method that allowed designers to identify Features in a geometric model and create the Feature model from the interpretation of the geometry. Initially, this was an interactive process that required the designer to analyse the geometry and interactively pick those entities in the geometric model that were to form a Feature.

Automatic Feature recognition is aimed at the execution of this process without any human interaction. Both types of Feature recognition resulted in Feature models that included information for application such as process planning and NC programming. In automatic Feature recognition a distinction is made between the recognition of machining regions and of predefined Features. Machining regions are, for example, the 2½D regions of a part that are removed by milling techniques. Although referred to as Feature recognition, the only characteristics of the Feature that are required for machining regions, are its boundaries. No distinction is made for machining regions between, for example, rectangular or L-shaped regions, because this is not necessary for the milling process. The machining regions identified in the geometry are not predefined in shape, only in the way they can be machined.

The recognition of predefined Features is different in that it compares the geometric model to the shape definitions of predefined Features in order to identify instances thereof in the geometry. Generally, the recognition process involves several phases: recognition of matching topologic and geometric patterns; determination of the parameters of the recognised Feature from the geometry; extraction of the recognised geometry from the model; completion of the Feature geometry; organisation of the recognised Features in a graph; and combination of simple Features into higher-level Features.

Features can be recognised from both boundary representations (B-rep) and volume-based (CSG) models, however, they require different kinds of recognition.
4.5 Approaches to Feature modelling

Approaches for which various techniques and algorithms have been developed. In the B-rep-based approaches the matching of graphs often plays a key role. The nodes in such a graph can be, for instance, the faces of the geometric model, while their relationships, the shared edges, form the arcs of the graph. Whether this relationship has a convex or concave nature appears to be another important factor in the recognition process. In the CSG-based approaches, the general goal is to rearrange the CSG tree in such a form that the CSG representations of the recognised Features can be separated from it. Among the major problems in these approaches is the fact that a CSG representation tree, although unambiguous, is not unique: the same eventual model can be represented by many different CSG trees, however, the Feature recognition process should lead to a unique arrangement of Features for all of these different representations.

4.5.2 Design by Features

Whereas the goal of Feature recognition approaches is to establish a Feature model from analysis of a given geometric model, the logical next step, also historically, has been to start the design by creating Features in the first place, rather than a purely geometric representation. The major advantage is that much of the design knowledge that is available during design but cannot be represented by geometric models, can now be accurately stored in the Feature model. The basis for this form of modelling can be the model of a raw stock from which the part is to be produced. An approach called destruction by machining Features involves subtraction of Features from the stock that correspond to the material removed by machining operations. Another approach, called synthesis by design Features, also allows addition of Features, so it is not necessary to start with a model of a raw stock.

The geometric model in the design by Features approaches is generated from the Feature model. Instances of Features are created from a library of Feature definitions. The instances are given values for their parameters, such as the dimensions and location, and related to the model as it is already built up, for example, a slot is related to the surface from which it is to be removed.

As mentioned before, constraints play an important role in Feature modelling. Two generic approaches for the management of constraints are distinguished on the basis of how the Features are represented. In procedural design by Features, the Features are represented in terms of rules and procedures for, e.g., instantiation, modification, deletion, derivation of parameters, and validation. As a result, the constraints in such a model are unidirectional, as is the possibility for propagation of changes. Another disadvantage is that procedures must be defined for every situation in which a Feature will be used.

In the approach called declarative design by Features, both the Features and the constraints are defined in a declarative manner. This allows these definitions to be more generic since they are more independent of the context. Change propagation in the declarative approach takes place by validation of the constraints after a change has been made to the model. The actual propagation only takes place once all constraints have
been satisfied, which can be an arduous process. Solving limited groups of constraints, independently of other constraints in the model, can enhance this process.

4.5.3 Feature conversion

When design Features are defined with the intention of explicitly facilitating later processes in the production cycle, this form of design is normally indicated as design for X, where X is, e.g., manufacture or assembly. In these cases, the specific requirements of these later stages, for example the available machining tools, stock, and assembly methods, have been taken into account, either in the definition of the design Features, or by definition of conversions from the design Features to the later process Features. The latter approach requires conversion mechanisms from one domain of Features to another. For example, a designer might define a slot Feature on a part, that in manufacturing stages is converted to a combination of two ribs. Because the focus in the mechanical engineering discipline is so strongly on the part’s shape, these conversion mechanisms are based on the interpretation and comparison of geometric descriptions of Features.

An example of this kind of research is that of de Kraker et al [1995, 1996], who address this problem by developing a method for multiple-way Feature conversion. One way Feature conversion only allows Features from early phases of the design and production cycle to be converted to subsequent phases. For concurrent engineering, however, phases coincide and require various Feature models of the same part to be maintained consistently. These various Feature models are called views, much the same as the views in product models as described in section 2.2.1. The conversion of Features from the various views is done through their geometric representation. For this purpose, the Feature model is represented at three levels: the Feature level containing the shape type and constraint data, both specified in an object oriented language; the canonical shape level representing the Feature as a parameterised solid (CSG tree); and the evaluated geometry level, which contains a so-called cellular model. The cellular model decomposes the Feature geometry of the part into smallest cells that each have a list of Features that their volume completely belongs to. This cellular model forms the basis for the conversion of Features from one view to another. Comparing again to the product modelling approaches as discussed in section 2.2.1, this cellular model of geometry fulfills the role of the core model in the product modelling approaches.

Crucial assumptions in this approach are that all information necessary to define a Feature in any of the views can be derived from the low-level geometry. This seems to be in contrast with the design by Features approach, which was devised to circumvent the problem of losing semantics when modelling in terms of geometry only, or by relying on Feature recognition from geometry. It is the strong bias on form Features that allows this approach to work successfully.
4.6 Actual research topics

Apart from the continuing research on the topics discussed in the previous section, two other actual topics in research on Feature-based modelling in mechanical engineering are relevant in the context of this thesis. The first is the integration of the design by Features and Feature recognition approaches into a single modelling system. The second addresses the issue of user-defined classes of Features.

The integration of the design by Features and Feature recognition approaches is expected to combine the advantages of both approaches and to result in design systems that will allow the designer to work either way: by starting with the geometry that possibly results from other modelling efforts, and working towards a Feature model; or by starting directly with the design in terms of Features and generating the geometry from the Feature model. Both ways of working should result in equivalent models that are represented and processed in a uniform manner. Moreover, the two ways of working should be allowed to concur, meaning that the designer can add pre-defined Features to a geometric model and that the automatic Feature recognition process is enhanced by the availability of such Features. The latter technique is called incremental Feature recognition, where the recognition process benefits from the Feature instances that are already present in the model [Laakko and Mäntylä 1993]. Examples of research projects that address the integration of design by Features and Feature recognition are found in [DeMartino et al. 1994] and in [Han and Requicha 1998].

The second topic of current research discussed here is user-defined Features. “Common experience [in mechanical engineering] has been that each new part gives rise to several new Features, in addition to the ones previously conceptualised.” [Hoffmann and Joan-Arinyo 1998] Shah and Mäntylä [1995, p.263] write that “the number of feature classes potentially needed in design cannot be limited. Therefore a system that restricts the user to a fixed set of predefined features is not likely to be useful for all design applications.” “New requirements are likely to lead to the introduction of new features. ... It is precisely this customisability that makes features powerful.” [ibid. p.98]

“Complete support of user-defined features in a design by features system requires that feature classes created by the user become full-privileged members of the feature collection of the system. That is, they can be created, deleted, and manipulated; they can have relationships to other features (and these relationships themselves can be defined too); they can be validated by validation constraints or rules (and new validation constraints and rules can be defined); their geometry can be anything that can be described in the underlying geometric modelling system. Clearly, to create a feature definition mechanism that covers all these facilities completely is a challenging task of software engineering that necessarily introduces certain complexity and overhead to the resulting system.” [ibid. p.165]

Chen et al. [1991] describe a framework for Feature-based part modelling. This framework consists of three main processes: (1) Feature creation, for the definition of Feature-classes; (2) Feature modelling, for instantiation of classes of Features into
generic Feature objects; and (3) part modelling, for placing the new Feature objects in the part model, for specification of Feature attributes, and for the addition of new Feature objects to a Feature relation graph. In the context of user-defined Features, especially the first process in this framework, Feature creation, is of interest. A Feature class is defined in the Feature creation process where the user:

- specifies its parametric shape;
- indicates the variable dimensions;
- selects the dimension type (defined during instantiation, or inherited from a parent);
- specifies orientation, origin, axes, et cetera;
- creates a Feature class (containing geometric definition and functional properties);
- and classifies the new class in relation with a predefined taxonomy in the Feature library.

A similar procedure for the definition of new Feature classes is proposed by Duan et al [1993], describing their Feature Solid-Modelling Tool (FSMT). This tool includes a Feature definition and management system that allows the definition of a Feature in three parts: (1) the description of the parametric geometry; (2) the attributes of the Feature (e.g. material, precision) and its relationships to other Features; and (3) the mapping from the definition into applications for design and manufacture and the knowledge related to the particular Feature, such as topological reasoning rules and consistency verification rules. Classification of new Feature classes is done on the basis of the so-called principle of function and geometry similarity.

The work of Hoffmann and Joan-Arinyo [1998] supports User-Defined Features (UDFs) as compound Features: Features that are compositions of standard Features provided by the modelling system or of other, previously defined UDFs. Such a user-defined composition serves as a larger, functionally meaningful design unit. The advantage of working with a user-defined composition, as compared to working with standard, simpler Features only, is that its behaviour can be predefined, for instance in relation to design changes, giving it more functional independence from other Features. Defining a UDF is a process of composition: one starts with a first Feature, either standard or other UDF, as the first component of a new UDF, continuing with the addition of other Features as components and of constraints and attributes. The definition procedure is concluded with the specification of an interface for the instantiation and usage of the UDF in the actual modelling process. Upon instantiation of a UDF, the user is requested to supply values for parameters of the Feature (in a non-fixed order), while other attributes are computed by way of constraints and equations. In-equations can be used in the definition of a UDF to specify valid ranges of values. The representation of UDFs by Hoffmann and Joan-Arinyo is done using the Editable representation (Erep) language that produces so-called generative Feature descriptions. This language provides mechanisms for the support of parameters, attributes, geometric and technological constraints, persistent naming, and variational constraint solving.
Bidarra et al. propose a declarative scheme for the definition of Feature classes [Bidarra et al. 1998]. This approach is a declarative approach employing various kinds of constraints to describe a Feature class. A UDF in this approach is defined as consisting of a shape definition and a number of constraints. The various kinds of constraints have the following functions:

- **attach constraints** specify how a Feature instance is attached to the model, by coupling some of its faces to faces of other Features already present in the model;
- **geometric constraints** specify geometric relations, such as parallelism and distance, between Feature faces;
- **dimension constraints** specify the set of values allowed for a Feature parameter;
- **algebraic constraints** specify an expression for Feature parameters;
- **semantic constraints** specify which topological variants of a Feature instance are allowed, by stating the extent to which the Feature's faces should be on the model boundary;
- **interaction constraints** specify whether a given interaction type should be disallowed for a Feature instance.” [ibid.]

Apart from geometric and topologic validity, these constraints are also used to check and maintain the functional validity that is not directly related to shape parameters but, for instance, to the volume or boundary of the Feature as a whole. Examples given by Bidarra et al. are the requirements that a subtracted Feature should not become a closed void inside the model nor should it be completely absorbed by another subtracted Feature.

![Image](image-url)
Feature-Based Modelling in Architecture

Using the requirements stated in chapter three as the determinants for the aimed development of an information modelling approach that supports architectural design, the area of Feature-based modelling is evaluated and the essentials extracted from the Feature concept for application in the architectural context. Modelling concepts developed in the architectural context that resemble the Feature concept are reviewed and evaluated in the light of the requirements. Subsequently, the analogy between the two fields, mechanical engineering and architecture, is examined more closely in order to identify the problems therewith. This leads to the formulation of additional principles that must underlie the design of the Feature-based modelling approach for architectural application. This chapter ends with a definition of the term 'Feature' in the architectural context.
5.1 The essentials of the Feature concept for architecture

The definition of Features in the original area of development of Feature technology in mechanical engineering is an important basis for the research project presented here. Since the focus in mechanical engineering is on describing shape and geometry, the definition of Form-Features as quoted in the previous chapter, needs to be considered. However, the architectural context in which the theory of Feature-based modelling will be developed is expected to require a more general definition of Features than one that is restricted to the shape or geometry of physical parts of a building.

Shah warns that too general a definition of the term Feature renders discussions on the subject meaningless [Shah 1991b]. He concentrates solely on the development of Form-Features. However, research in the area of Product Modelling in the context of Building & Construction, as discussed in section 2.1, shows that geometry or shape cannot be the basis for an adequate approach to modelling the information concerning a building throughout its life-cycle, including the design stages. Therefore, from the definitions used in literature from the mechanical engineering context, a generalisation of the concepts of Feature-based modelling is distilled. These generalised concepts are compared to existing applications of Feature technology and similar concepts in architecture in the next section. With this knowledge and the requirements as stated in chapter 3, a definition of the term Feature in the architectural context is given in section 5.5.

Concentrating research on the shape of products, Shah contemplates the definition of a Form-Feature. A purely geometric modelling definition does not suffice, since this “does not include the reason for a Feature’s existence or usefulness. Because Features encode the engineering significance of the geometry, we extend the definition to include the purpose for which a Feature is useful. Features are generic shapes with which engineers associate certain properties or attributes and knowledge useful in reasoning about the product. Features can perhaps be thought of as Engineering Primitives suited to some engineering task. One can use them as building blocks for product definition or for geometric reasoning.” [Shah 1991b]

The concepts in the Feature-based approach that are of interest to the architectural context can be summarised in the following issues:

- **Modelling the rationale of a design.**
  This involves the representation of the semantics of the design, its meaning in terms of function and behaviour of parts of a design, how parts can be produced, their properties during evaluation, et cetera.

- **Using Features as building blocks or primitives for defining and reasoning about a product.**
  Features are the formalisation of design terminology. Building information models using Features can therefore eliminate the step of translating design...
decisions into data definitions that do not naturally belong to the domain of design. Consequently, Feature models are an accurate reflection of the design rationale.

- **The autonomous definition of Features.**
  The definition of a Feature is relatively independent of its context, leading to a more autonomous function of the Feature in the information model. This is manifested in its behaviour in relation to the model, as well as in the way the information model is generated during design: as a composition of Features, not in a rigidly predefined structure, but in a flexible network of relationships that are established during the process of design.

- **The formalisation of design concepts in Feature definitions.**
  As design concepts are established or newly created, Feature definitions can be formally specified to reflect the new entities in the conceptual information model of the domain of design. This way, the tools for design can be adapted to the specific needs of each individual design case, enlarging the library of available resources for design.

The essence of the design by Features approach in an architectural context lies in its function towards the dynamic behaviour of design models. The issues listed above form the basis for the theory on modelling architectural design information, developed in this thesis, that fulfills the requirements of flexibility and extensibility as stated in chapter 3.

### 5.2 Features and similar concepts in architecture

Some concepts that have been developed and used in the research area of modelling design information and knowledge are related to the issues mentioned above. The term *feature* also appears in some research areas, though not always with the same intention, and not always under the caption of *feature*. Six of these areas of research are discussed here in brief, indicating their relation to the Feature concept, or how the term *feature* is used in their context.

**The GARM**

The General AEC Reference Model (GARM) developed by Gielingh [1988], is often regarded as the first attempt to define a core model for the AEC industry (see also section 2.3). This model defined a meta-model that was to form the basis for the definition of application specific models in the different disciplines of AEC. The meta-model is based on one generic entity which represents a product, or a decomposition of the product. This generic entity is called Product Definition Unit (PDU) and has characteristics which refer to certain aspects of the product. Different levels of decomposition of a product are distinguished in the GARM, a PDU is the basic entity for each of these levels. A PDU can be a system, a sub-system, a component, a part, a
feature, a space, or a joint. These are generic entities of the decomposition of a product, the particular decomposition is to be specified by the designer of a particular product.

A feature, in the context of the GARM, is defined as the Form-Features in mechanical engineering: an area of interest on the surface of a part. The concept of the PDU as a generic entity in decomposition structures may be of greater interest with respect to the requirements of flexibility of information models. However, the kind of model that is proposed by the methodology of the GARM is characterised by a strictly hierarchical composition. There is no provision for adaptation of the model to changing requirements during the design course, which means that this kind of model will be too rigid for use in the dynamic context of design stages.

**Modes and features**

In [Lawson and Roberts 1991], an analysis of data in early phases of design is presented, showing four structurally different ways of organising knowledge about an emerging building design, which are in common usage. These different organisations of knowledge, or modes of thinking about a design, are related to characteristics (called features) of the design in these different types of organisations. Examples of these modes of thinking about design and the features that are related to them are given:

<table>
<thead>
<tr>
<th>modes</th>
<th>features related to this mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>component</td>
<td>wall, window, door, floor, ceiling, roof</td>
</tr>
<tr>
<td>envelope</td>
<td>space, courtyard, external skin</td>
</tr>
<tr>
<td>stratum</td>
<td>floor level, section, elevation</td>
</tr>
<tr>
<td>system</td>
<td>structure, cladding, services, circulation</td>
</tr>
</tbody>
</table>

The term *feature* is used here with the meaning of “a characteristic of an architectural artefact”. Lawson and Roberts state that CAD support in design should not make any assumptions on the sequence of input or the types of data. Instead it should be able to interact with the designer at least in all of the mentioned modes and establish data-structures adequate for each of these modes. They recognise that all these modes of design are mutually interdependent. Architects, during the course of design, constantly “flip from one mode to another in quick succession”. Also, different evaluation tools will require different kinds of input data, which must be consistently provided from any of the modes. Therefore, CAD systems need to be able to make inferences across modes, for instance by representing features which are input or manipulated in one mode, by features in other modes.

What can be learned from the approach to organising data in early design stages by Lawson and Roberts, is that features in their conception, although not explicitly defined, are general characteristics of architectural artefacts. These features are strongly related to the way of thinking about the design, the modes, while the domains of these
modes are very likely to overlap. This means that information on a design is collated in features with a certain level of redundancy, which requires special attention in terms of consistency of the overall model. Also, features are strongly related to each other and especially the relationship between features of different modes is an important issue for CAD systems to support.

**Features in product modelling for AEC**

Features are found in other research projects on product modelling, but always from the perspective of Form-Features, applied for the description mainly of elements of steel constructions. An example of this is the CIMsteel project which defines the Logical Product Model (LPM), or Product Model for Structural Steelwork [Watson 1994]. This model has been based on the techniques deriving from the developments in STEP, and has itself been the basis for the specification of the application protocol AP 230 Building Structural Frame: Steelwork [Ward 1994]. Features in this model are simple Form-Features used only in the detailing phases of design. They include Features such as chamfer, notch, hole, and skewed end. Other operation-like characteristics of parts, such as welds, are not defined using the term Feature.

**The BAS-CAAD project**

The conceptual model proposed in the BAS-CAAD project, introduced already in section 3.5, contains generic concepts that evolve during the course of design by an incremental determination of their properties. These generic concepts, which Ekholm and Fridqvist [1996] at early design stages call things or, when they have composition and internal structure, systems, are used to specify the characteristics of buildings and spaces, as well as the properties and activities of human social behaviour. As design proceeds, these generic concepts are successively specified into detailed schemata for organisations, buildings, and spaces.

Ekholm and Fridqvist argue that it is essential to model the organisation and behaviour of users of a building, the activities in the building, independently of the building model itself. This allows user characteristics to have their own representations, which is especially useful in early design stages like the briefing stage. Also will it allow more feedback of information about the impact of different building solutions on the user activities.

---

1 This notion of overlapping concepts in design is also discussed by Alexander [1965]. Alexander defines a unit as a collection of elements in a city that relate to each other, for instance a traffic light, a sidewalk, and a newsrack next to the traffic light. Alexander argues that units in a city are neither contained in each other, nor mutually exclusive: they overlap. Therefore, a city cannot be represented by a tree-like structure, but rather requires a semi-lattice structure to allow for its units (or concepts of design) to overlap.
**The Engineering Data Model**

In the EDM-2 project, also introduced before in section 3.5, defines so-called Design Entities. A Design Entity (DE) consists of (1) a set of attributes that are defined within the DE and that have a domain or class defined by another DE, and (2) a set of calls to constraints that have the DE as their scope. DEs support inheritance in two ways in the EDM-2: inheritance by specialisation (this is what is called inheritance in OO approaches) and inheritance by subsumption (in OO this would be called decomposition). Compositions of DEs are used to define, for instance, assemblies or collections of DEs that are to be used in a particular application. Compositions in the EDM-2 are different from the aggregations normally found in OO approaches. A composition in EDM-2 does not result in a new, composite entity, but rather specifies a relationship over a set of DEs. This set of related DEs, the parts, can optionally be related to another DE, the target. However, this relationship is defined outside the definition of this target DE: the target entity is not modified for this additional relationship. The following example, taken from [Eastman and Jeng 1998], illustrates this.

```plaintext
CREATE DE wall
  ATTR (area: sq_ft, geometry: solid, const: const_method)
  DESC "a simple wall";

CREATE COMP wall_assembly
  TARGET wall
  PART (window, door)
  CCAL not_overlapping CONSTRAINT b_box (part_window, part_door): V
  MAPCALL comp_area MAP derive_area (part_window, part_door, tar_wall)
    RETURN (tar_wall.area)
  DESC "a composition for a simple wall";
```

**The Semantic Modeling Extension**

The SME system by Clayton et al. [1999], which also was introduced already in section 3.5, associates features with the graphic entities in a CAD drawing. These features represent the function of the associated form and are used by evaluation modules in the system to predict the design's behaviour. The feature's definition includes the attributes and functionality necessary for the association with the graphic entities in the CAD drawing, and stores the functions and materials that are associated with the feature in the context of the evaluation area it is programmed for. Features in the SME system are defined under the responsibility of software developers that provide the evaluation modules that designers can use within this system.

**5.3 Problems for the analogy between the fields of application**

The methods and techniques for modelling Features in mechanical engineering cannot be simply transferred to the area of architectural design. The analogy between the two fields of information modelling does not hold on some points, which will be discussed in...
this section. In the next section, this leads to the statement of additional principles for the application of Feature modelling in the architectural context.

**Dealing with complexity and hierarchical information structures**

The first difference between the two areas of design concerns the complexity of the product under design, and the way this is dealt with during the course of design. The Feature-based modelling approach in the area of mechanical engineering is characterised by its focus on describing the shape and geometry of parts of a product. The complexity and composition of the product are not modelled directly by means of Features. Some Features address the assembly of parts, but Features are not used to define characteristics at higher levels in the hierarchy of a product's description. The reason for this strict, hierarchical decomposition of products into parts and Features (see figure 4-2 on page 64), lies in the fact that products in mechanical engineering are relatively well-defined and of a small scale, as compared to buildings. Because of this characteristic of products being well-defined, the design and detailing of their parts can be separated from the context. The requirement specifications of the parts are assumed to be available before detailing is started by means of adding Features. The relationships of parts and Features are adequate for the specification of how requirements can be fulfilled in the detailed design of the part.

In the architectural design context such a clear hierarchy cannot be defined for the requirement specifications, the characteristics of the building and its parts, and their relationships. Requirements are specified in a way that does not logically define the hierarchy of the building that is to fulfil these requirements. Consequently, the design process cannot follow a hierarchical approach, strictly top-down, for the specification of the building's composition. The composition of the building is incrementally formed during the process of design, while the properties of the building are related to many different levels of abstraction in this composition, e.g. to the building as a whole, to sections of the building, elements, details, and so on. As discussed in chapter 3, dynamic design processes in architecture involve a constant switching between levels of detail and abstraction in design, relating information between these levels, restating problems, and dynamically finding solutions. The information generated during this kind of design process requires a structure that is not a strict hierarchy, and especially not limited to detailed levels of composition only. As Lawson and Roberts [1991] recognise, the design rationale will consist of many different modes of thinking about the resulting building. These modes overlap and cannot be captured in a hierarchy.

**Abstract, non-physical concepts**

A second difference between mechanical engineering and architectural design that is important for the organisation of information models concerns the kind of information that is dealt with and, perhaps more important, how this information is dealt with. From a rather general point of view, mechanical engineering can be characterised as concerning purely technical design problems and dealing with physical aspects of the product, at least so at the stages of detailed design in which FBM plays a role. In contrast,
architectural design is very much concerned also with solving problems of a social, cultural, and human order. Aesthetics and how a building is experienced, for instance, play a very important role, even in the more technical and detailed areas of architectural design. Consequently, the kind of information that is dealt with in architectural design may be of a more abstract nature. Concepts and variables of architectural design are not necessarily related to physical and concrete objects of design and are often not measurable. Examples of such concepts are space, user-function, costs, safety, comfort, circulation, escape-routes, et cetera.

5.4 Additional principles for architectural Feature-based modelling

The differences between the fields of design in mechanical engineering and architecture, as discussed above, indicate that for the Feature modelling technique to be applicable in an architectural context, some adaptation is required. Three additional principles are added to the Feature-based modelling paradigm for its development in the architectural context [van Leeuwen, Wagter, and Oxman 1995, 1996]. Where design in mechanical engineering can be characterised as a top-down approach with Features used only in the most detailed phases, the architectural design process does not necessarily follow a top-down approach and requires a stronger integration of information modelling at various levels of abstraction. The modelling approach in architecture must address the complex domain of architectural design in its entirety, dealing with both space and matter as concepts in design.

1. **Subject of the modelling activities is the product as a whole.**
   Where in mechanical engineering a single part of a product is the subject of a Feature model, in architecture the building in its entirety is subject of the information model. This means that the information involved represents the building in full complexity, including the relationships between different levels of detail and decomposition.

2. **Features are attached to multiple levels of abstraction.**
   As a consequence of the above, information in the model will be related to any level of abstraction of the building, this includes the detailed levels as well as the more general levels concerning, e.g., parts of the building or the building as a whole. Modelling this information using Features for purposes of extensibility and flexibility requires that Features can be related to multiple levels of abstraction.

3. **Features represent concepts that can be non-physical as well as physical.**
   Physical concepts are those that have a direct relation to any part or element in the building as it will be physically present when completed. Non-physical concepts include all those that cannot be directly related to tangible or visible objects. In early stages of a design process this sort of non-physical concepts is
commonly present. But also during detailed stages of design and in the construction process, information not related to physical objects remains relevant and needs to be modelled. Hence, Features will be representing both physical and non-physical concepts in building models.

5.5 Definition of the term Feature in the architectural context

Summarising from chapter 3 and the sections 5.1 to 5.4, the approach to modelling architectural information that this research project aims to develop is based on:

1. the requirement that information models for the support of design should evolve with the dynamic nature of design, meaning that such a model should provide sufficient extensibility and flexibility (discussed in chapter 3);
2. the concepts from FBM in its original field of application (mechanical engineering) that are considered relevant for the field of architectural design (discussed in section 5.1):
   a. modelling the rationale of a design;
   b. using Features as building blocks or primitives for defining and reasoning about a product;
   c. the autonomous definition of Features;
   d. the formalisation of design concepts in Feature definitions;
3. the additional principles for application of the FBM approach in the area of architectural design as discussed in section 5.4:
   a. subject of modelling activities is the product (the design) as a whole;
   b. Features are attached to multiple levels of abstraction;
   c. Features represent concepts that can be non-physical as well as physical.

A definition of the term Feature in the context of architectural design must be based on these concepts and requirements. Starting from the concepts in FBM, in particular points 2.a and 2.b above, a Feature will represent the rationale of a design and forms a unit of information that can be used in reasoning about a design. This means that, to the designer, Features represent themselves using the terminology that is used in the designer’s discipline and therefore suits the context of the design:

*A Feature is a collection of information with semantic meaning to a designer.*

From the FBM concepts 2.b and 2.d follows that a Feature is a collection of information that forms a coherent entity in the information structure of a design model and is the formalisation of a design concept:

*A Feature is a coherent formalisation of a design concept.*

As a consequence of the way designers handle design concepts, which often involves relating and organising them in an ad hoc manner, the formalisation of design concepts, the Features, must have an autonomous definition (2.c). This means that the
dependencies between Features should be minimised and their inter-relationships be as flexible as possible:

A Feature is an autonomous entity of information.

From the principles for application of the FBM approach in the architectural design context (point 3 above and section 5.4) follows that an architectural Feature model may represent a complete building design and, as a consequence, that Features represent information related to many different levels of abstraction. Also, the definition of Features is not restricted to physical components of a building, but may refer to more abstract, i.e. non-physical, concepts as well. Therefore:

Features represent concepts, either physical or non-physical, at various levels of abstraction, composing a building model.

Finally, returning to point 1 at the beginning of this section, there are the requirements of extensibility and flexibility for information models in architectural design. The flexibility of a Feature model is ensured by the autonomous character of Features, which renders them independent of their context and allows modelling flexible relationships between design concepts. However, the extensibility is not yet clearly covered by the definition of the term Feature. Extensibility means that the definition of a Feature cannot be anticipated but may occur at any point in the process of design and modelling. In other words:

The definition of a Feature possibly emerges during design.

These many aspects of the notion ‘Feature’ as perceived in this work combine into the following definition of the term ‘Feature’ to be used in the context of modelling architectural design information:

A Feature is an autonomous, coherent collection of information, with semantic meaning to a designer and possibly emerging during design, that is defined to formalise a design concept at any level of abstraction, either physical or non-physical, as part of a building model.

Examples of architectural Features are: building elements, such as walls, floor-elements, window frames, hinges, and outlets; materials, such as concrete, plywood, and aluminium; spatial functions, like living, cooking, drawing, and reading; structural functions, such as bearing, supporting, stabilising; physical properties, including colour, strength, elasticity; non-physical characteristics, like costs, maintainability, maintenance sensitivity. The next chapter deals with an inventory of what kind of information is modelled in architectural design, and thus what kind of Features are to be expected in design models.
5.6 Structure of a Feature model

The above definition of the term Feature has a number of consequences for the structure of design information models that are based on Features. According to the definition, Features are the primitives in these models, they formalise the design concepts and represent information at multiple levels of abstraction. To satisfy the requirement that designers wish to handle information concerning design concepts in a very flexible manner, Features are defined as autonomous entities of information.

To retrieve an insight in the structure of the Feature model, the structure of the Features must be determined. The structure of a Feature can be regarded as a model in itself, where the characteristics of the concept are the information entities, or indeed formalised concepts themselves. This decomposition of a concept continues to the point where a concept no longer has characteristics other than a single value. Each of the concepts in this sequence of decompositions is formalised as a Feature that is related into the composition of its parent Feature as one of its characteristics, and has a collection of relations to other Features that represent its own characteristics. The content of the design information model is thus constructed from a network of Features, where Features represent both design concepts and the characteristics of design concepts, which are concepts themselves at another abstraction level.

Characteristics can be shared by multiple concepts, for instance, a door and its frame may share the same colour characteristic. Therefore, the network of Features is not a tree-like network, which would be the case if the door and the frame each had their own definition of the colour characteristic, but rather a lattice-like network or a graph. The nodes in the graph are the Features, the arcs are the relationships between the Features. Since the relationships are directed from one Feature to another, from the concept to its characteristic, the graph is a directed graph. This does not imply that the reverse relationships cannot be used in reasoning with the model.

The kind of relationships that can exist between Features is not limited to decomposition relationships. An example of another kind of relationship is the association, like the association of the door with the frame. The various kinds of relationships, as well as the various kinds of Features that can exist in a Feature model,
are defined and discussed in detail in chapter 7 which contains the formal specifications of the Feature-based modelling approach for architectural design.
An Inventory of Feature Types in Architecture

In the previous chapters, the requirements and principles for modelling architectural Features have been determined and the term ‘Feature’ in the architectural context is defined. Before an appropriate modelling approach with Features can be developed, the objective in this chapter must be to obtain sufficient insight in the extent and complexity of architectural information that will be modelled with this approach. For this purpose, various sources of architectural information are examined, including national and international classifications of building elements, documentation systems for building products and materials, and representative conceptual product models. Selections of these sources are studied first in general, determining the kind of content and structure they manifest, then more deeply in a case study using a chosen subset of information, namely vertical space boundaries. From this case study, conclusions are drawn in relation to what kind of Features can be expected in a model for vertical space boundaries. At the end of this chapter, these conclusions are generalised into a categorisation of types of Features that can be expected to emerge with the FBM approach in architectural design.
6.1 Getting an insight in the extent and complexity of architectural information

Before the methodology of modelling architectural information on the basis of Feature technology can be further developed, it is necessary to get a deeper insight in the information and type of information that is related to buildings and systems in buildings. This insight is acquired through the analysis of various sources of architectural information used in practice and research on architectural design and construction. The analysis is illustrated with a small case study on a restricted set of architectural information, indicating the extent and complexity of information related to vertical space boundaries. This analysis leads to an inventory of the kind of information that is related to vertical space boundaries, and results in a set of conclusions towards the kind of Feature types that will be needed to represent this information. The inventory is used in section 6.7 to categorise types of Features. Figure 6-1 shows the described methodology for the analysis of architectural information on which the categories of Feature types are based.

**Analysed sources of architectural information**

In order to get an insight in the information and type of information that is related to buildings and systems in buildings, several groups of sources can be analysed. The first group of sources is formed by the various national and international classifications of building elements or parts. A review of these classifications in section 6.2 delivers an overview of the physical parts distinguished in buildings and of the functions to be performed by buildings and systems in buildings.

A second source for the inventory of architectural information is found in the vast amounts of documentation on building products and materials providing technical details, performance characteristics, usage guidelines, et cetera. The review in section 6.3 provides an overview of the technical aspects that can be related to building elements and parts.

The third source to consider is found in the building codes for regulations on the performance of buildings and their systems. This information can be analysed in conjunction with documentation of design
methods in, for instance, disciplines of structural design and HVAC engineering. The kind of information that results from a review of this type of sources concentrates on the terminology and concepts used in processes of design and evaluation in these particular domains. The complexity and extensiveness of such a review is enormous, even for small and well-confined disciplines; a case study that demonstrates this complexity is found in the work of Tacke [1994]. A review of this kind goes beyond the scope of this thesis.

A more accessible way of gathering this kind of information, albeit more limited as well, is looking at the results of information modelling in the area of product modelling, the fourth source. Disregarding the problems that are involved in applying product modelling techniques in design stages as discussed in section 2.4, the contents of the resulting models can contribute substantially to an in depth understanding of the full range of information in the Building & Construction industry. It is recognised that research results from the area of product modelling do not represent the full depth of information needs in the various disciplines for which product models have been developed. However, the models reviewed in section 6.4 can be regarded as representative for the most relevant information needs in the domain for which they have been developed.

6.2 Classifications of parts and elements

Classification of information in the Building & Construction industry has been subject of both national and international standardisation bodies over the past several decades. Classifications of construction elements have been defined in national standards such as the Dutch NL-SfB or Elementenmethode '91 [SBK 1991] and the British CI SfB-1 Elements classification [Mills and MacCann 1968], both of which are based on the original Swedish SfB classification. The German DIN 276 Kosten im Hochbau (see [Keller 1986]) also classifies building elements, principally for the purpose of cost calculation, as does the Dutch STABU methodology for specification writing. The latter is not solely based on building elements, but classifies activities, means, tools, and results in the construction process as well. Some examples from the STABU table are given in table 6.1 (see [Woestenenk 1997] and [STABU 1995]). Examples of the categories in the NL-SfB are included in the case study in section 6.5.

<table>
<thead>
<tr>
<th>Notation</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>B200000</td>
<td>basic products</td>
</tr>
<tr>
<td>B210000</td>
<td>blocks, plates/sheets, tiles (stiff)</td>
</tr>
<tr>
<td>B211212</td>
<td>clay bricks</td>
</tr>
<tr>
<td>B211610</td>
<td>glass masonry blocks</td>
</tr>
</tbody>
</table>

1 SfB stands for Samarbetsskommittén för Byggnadsfrågor, the Swedish committee for standardisation in the Building & Construction industry.
A very valuable source for an inventory and classification of architectural information is the work done in Working Group 3 of the International Construction Information Society (ICIS WG 3). The aim of this working group is to develop an international classification table for Elements. Elements in this context are defined in the ISO Technical Report TR 14177 “Classification of information in the construction industry” [ISO TC59 1993]:

An element is a physical part or system of a facility with a characteristic function (e.g. enclosing, furnishing or servicing building spaces), defined without regard to the type of technical solution or the method or form of construction.

Discussion continues within this group whether a distinction should be made between elements that are the physical parts in the construction process and elements that are defined by functions to be fulfilled by the building. One document from Working Group 3 of ICIS is the proposal by Woestenenk [1995] for international conversion tables for parts and functions. This document distinguishes the two kinds of elements mentioned above, calling them parts (for physical parts) and elements (for functions) respectively. It lists as many as 15 national elements classifications and the table for product functions developed by the European Product Information Co-operation (EPIC), which also developed the table for Product Grouping. The conversion tables are an attempt to find common denominators for all elements in the different standards, in order to relate them in the manner as indicated in figure 6-2.
Figure 6.2 Conversion tables for parts and functions in ICIS WG3.

Some examples of these relations are listed in the following tables.

**Table 6.2 Analyzed source: Related parts in different national classifications and the ICIS WG3 proposed conversion tables.**

<table>
<thead>
<tr>
<th>entry in ICIS WG3 conversion tables</th>
<th>related entries in national classification standards</th>
</tr>
</thead>
<tbody>
<tr>
<td>building, construction: floor</td>
<td>EPIC Functions: structural floor components</td>
</tr>
<tr>
<td></td>
<td>German DIN 276: floors and ceilings, floor structures</td>
</tr>
<tr>
<td></td>
<td>NL-SfB: floors; structural, floors; non structural, floor finishes</td>
</tr>
<tr>
<td>building, equipment: heating</td>
<td>EPIC Functions: local heating units, electric heating terminals, storage heaters, convector radiators</td>
</tr>
<tr>
<td></td>
<td>Danish standard: space heating services</td>
</tr>
<tr>
<td></td>
<td>NL-SfB: heat generation, local</td>
</tr>
<tr>
<td>building, system: heating</td>
<td>German DIN 276: heat supply systems</td>
</tr>
<tr>
<td></td>
<td>British CI SfB-1: space heating, British BCIS Bldg heating installations</td>
</tr>
<tr>
<td></td>
<td>NL-SfB: heat generation, supplied heat, heat generation, total energy systems</td>
</tr>
</tbody>
</table>
### Table 6.3 Analysed source: Function – part relationships in the ICIS WG3 proposed conversion tables.

<table>
<thead>
<tr>
<th>function in ICIS WG3 conversion tables</th>
<th>related parts</th>
</tr>
</thead>
<tbody>
<tr>
<td>physical function: structural: spanning</td>
<td>• bridge</td>
</tr>
<tr>
<td></td>
<td>• building, construction: beam</td>
</tr>
<tr>
<td></td>
<td>• building, construction: floor</td>
</tr>
<tr>
<td></td>
<td>• building, construction: rafter</td>
</tr>
<tr>
<td></td>
<td>• building, construction: roof</td>
</tr>
<tr>
<td></td>
<td>• building, construction: wall, lintel</td>
</tr>
<tr>
<td>space function: access</td>
<td>• building, construction: floor, step</td>
</tr>
<tr>
<td></td>
<td>• building, construction: space</td>
</tr>
<tr>
<td></td>
<td>• building, construction: staircase/ramp</td>
</tr>
<tr>
<td>space function: space division</td>
<td>• building, construction: partition</td>
</tr>
<tr>
<td></td>
<td>• building, construction: partition, door/window</td>
</tr>
<tr>
<td></td>
<td>• building, construction: wall</td>
</tr>
<tr>
<td></td>
<td>• building, construction: wall, window</td>
</tr>
<tr>
<td></td>
<td>• building, construction: wall, window, equipment</td>
</tr>
<tr>
<td></td>
<td>• building, construction: floor</td>
</tr>
<tr>
<td></td>
<td>• building, construction: roof</td>
</tr>
<tr>
<td></td>
<td>• building, construction: ceiling</td>
</tr>
<tr>
<td>space function: climate condition</td>
<td>• building, system: air treatment</td>
</tr>
<tr>
<td></td>
<td>• building, system: heating</td>
</tr>
<tr>
<td>space function: communication: audio</td>
<td>• building, system: communication: audio</td>
</tr>
<tr>
<td></td>
<td>• building, system: communication: conference</td>
</tr>
<tr>
<td>space function: communication: image</td>
<td>• building, system: communication: display</td>
</tr>
<tr>
<td></td>
<td>• building, system: communication: image</td>
</tr>
<tr>
<td></td>
<td>• building, system: communication: image, equipment</td>
</tr>
<tr>
<td>user function</td>
<td>• building, equipment</td>
</tr>
<tr>
<td></td>
<td>• building, equipment: furniture</td>
</tr>
<tr>
<td></td>
<td>• building, equipment: signalisation</td>
</tr>
<tr>
<td></td>
<td>• building, equipment: technical</td>
</tr>
<tr>
<td></td>
<td>• building, system: lighting</td>
</tr>
<tr>
<td>user function: body care: washing</td>
<td>• building, equipment: sanitary</td>
</tr>
</tbody>
</table>

Parts and functions in the ICIS WG3 tables have been given assignments: a part assignment, based on decomposition and denoted with a comma (,), and a function assignment, based on specialisation and denoted with a colon (:). These assignments are called expressions that consist of combinations of terms. For instance the assignment ‘space function: supply: energy’ is the expression that specifies the function of energy supply to a space. The assignment ‘building, system: heating, conduit’ first decomposes the building into systems and then specifies the heating system which in turn is decomposed into a conduit.
Limited semantic content in classifications

The examples above provide an indication of the general approach in national classifications of information in the construction industry. The level of detail to which information is classified is generally restricted to the identification of the physical parts found in the building and its systems, and the identification of functions to be performed by the building and its systems. The actual information that is related to the characteristics of these parts and functions is not within the scope of these classifications. Therefore, the level of semantic content that can be communicated on the basis of these classifications is restricted to the identification of the different parts and functions, it does not include the semantics of, for instance, technical aspects and behaviour of the parts and elements.

6.3 Product- and material documentation

Another way of getting a deeper insight in the content and complexity of information in the construction industry is looking at documentation of construction products and materials. Generally, this information is available from manufacturers, therefore it should be noted that it is the manufacturer's point of view that is presented in this type of documentation.

In the Netherlands, a catalogue of construction products and materials is being maintained and published by the NBD (Dutch Construction Documentation) [1997]. This catalogue is a source commonly used in both design and construction phases. It documents the actual performance specifications of products and materials available on the Dutch construction-market. The information is organised using the NL-SfB classification for building parts and materials. Per product or group of products it provides manufacturer-supplied information, both graphical and textual. The textual information is given in a fixed set of categories. A review of these categories will assist in acquiring an overview of relevant information in the construction industry. Table 6.4 shows a summary of the information categories. They are labelled A-Z, though not all letters are currently being used.

<table>
<thead>
<tr>
<th>cat.</th>
<th>contained information in the NBD documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>short description</td>
</tr>
<tr>
<td></td>
<td>firm, brand, type, product-name, typical characteristics and differences between types</td>
</tr>
<tr>
<td>D</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>composition</td>
</tr>
<tr>
<td></td>
<td>system-composition (composition, material, appearance, properties, applications, etc.)</td>
</tr>
<tr>
<td></td>
<td>element-composition (components, materials, etc.)</td>
</tr>
<tr>
<td></td>
<td>material (types, brands, qualities, physical properties)</td>
</tr>
<tr>
<td></td>
<td>method of fabrication</td>
</tr>
<tr>
<td></td>
<td>surface-treatment</td>
</tr>
<tr>
<td></td>
<td>accessories</td>
</tr>
<tr>
<td>cat.</td>
<td>contained information in the NBD documentation</td>
</tr>
<tr>
<td>------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>F</td>
<td>shape, measures, weight</td>
</tr>
<tr>
<td></td>
<td>shape (main shape, illustrations, patterns, consistency)</td>
</tr>
<tr>
<td></td>
<td>dimensions (main dimensions, real dimensions and working dimensions, tolerances, modularity)</td>
</tr>
<tr>
<td></td>
<td>weight (whole product or parts, mass)</td>
</tr>
<tr>
<td>G</td>
<td>appearance</td>
</tr>
<tr>
<td></td>
<td>surface-structure</td>
</tr>
<tr>
<td></td>
<td>colour</td>
</tr>
<tr>
<td></td>
<td>shininess</td>
</tr>
<tr>
<td></td>
<td>side-effects (smell, taste, noise)</td>
</tr>
<tr>
<td>H</td>
<td>mechanical properties</td>
</tr>
<tr>
<td></td>
<td>product-strength (including stability, permissible loads, moment of resistance and inertia, etc.)</td>
</tr>
<tr>
<td></td>
<td>material-strength (tensile strength, elasticity, hardness, wear resistance, etc.)</td>
</tr>
<tr>
<td></td>
<td>surface-properties (friction, durability, scratch resistance, etc.)</td>
</tr>
<tr>
<td></td>
<td>dynamic properties (flexibility, shock-proofness)</td>
</tr>
<tr>
<td>J</td>
<td>fire, explosion</td>
</tr>
<tr>
<td></td>
<td>combustibility</td>
</tr>
<tr>
<td></td>
<td>inflammability</td>
</tr>
<tr>
<td></td>
<td>fire-propagation</td>
</tr>
<tr>
<td></td>
<td>smoke-development</td>
</tr>
<tr>
<td></td>
<td>fire-resistance</td>
</tr>
<tr>
<td></td>
<td>behaviour in case of fire</td>
</tr>
<tr>
<td></td>
<td>explosion-risk</td>
</tr>
<tr>
<td></td>
<td>residues after combustion</td>
</tr>
<tr>
<td>L</td>
<td>gases, liquids, solids</td>
</tr>
<tr>
<td></td>
<td>air-transmission</td>
</tr>
<tr>
<td></td>
<td>water-tightness</td>
</tr>
<tr>
<td></td>
<td>diffusion (vapour-permeability)</td>
</tr>
<tr>
<td></td>
<td>humidity-absorption</td>
</tr>
<tr>
<td></td>
<td>solubility</td>
</tr>
<tr>
<td></td>
<td>compoundness</td>
</tr>
<tr>
<td></td>
<td>dehydration</td>
</tr>
<tr>
<td></td>
<td>resistance (corrosion, erosion, moulds, etc.)</td>
</tr>
<tr>
<td>M</td>
<td>thermal properties</td>
</tr>
<tr>
<td></td>
<td>dilatation (linear, cubic)</td>
</tr>
<tr>
<td></td>
<td>conductivity (heat-conduction, heat-transmission, heat-resistance)</td>
</tr>
<tr>
<td></td>
<td>radiation</td>
</tr>
<tr>
<td></td>
<td>convection</td>
</tr>
<tr>
<td></td>
<td>absorption (solar energy transmission)</td>
</tr>
<tr>
<td></td>
<td>usage-temperature</td>
</tr>
<tr>
<td></td>
<td>resistance (melting point, boiling point, permissible temperatures)</td>
</tr>
<tr>
<td>N</td>
<td>optical properties</td>
</tr>
<tr>
<td></td>
<td>colourfastness</td>
</tr>
<tr>
<td></td>
<td>reflection power</td>
</tr>
<tr>
<td></td>
<td>transparency</td>
</tr>
<tr>
<td>O</td>
<td>acoustic properties</td>
</tr>
<tr>
<td></td>
<td>sound-absorption</td>
</tr>
<tr>
<td></td>
<td>sound-production</td>
</tr>
<tr>
<td></td>
<td>sound-production</td>
</tr>
<tr>
<td></td>
<td>-</td>
</tr>
<tr>
<td>P</td>
<td>acoustic properties</td>
</tr>
<tr>
<td></td>
<td>sound-production</td>
</tr>
<tr>
<td>Q</td>
<td>electricity, magnetism, radiation</td>
</tr>
<tr>
<td></td>
<td>grounding</td>
</tr>
<tr>
<td></td>
<td>lightning conductivity</td>
</tr>
<tr>
<td></td>
<td>cathodic security</td>
</tr>
<tr>
<td>R</td>
<td>energy and other factors</td>
</tr>
<tr>
<td></td>
<td>capacity (charge)</td>
</tr>
<tr>
<td></td>
<td>connection-values (tension, phase, frequency, currency)</td>
</tr>
<tr>
<td></td>
<td>power-consumption</td>
</tr>
<tr>
<td></td>
<td>power-production</td>
</tr>
<tr>
<td></td>
<td>material-consumption</td>
</tr>
<tr>
<td></td>
<td>fuel-consumption</td>
</tr>
<tr>
<td></td>
<td>water-consumption</td>
</tr>
<tr>
<td></td>
<td>side-effects (to application)</td>
</tr>
<tr>
<td></td>
<td>compatibility (to other materials, potential-differences)</td>
</tr>
<tr>
<td></td>
<td>durability</td>
</tr>
<tr>
<td></td>
<td>penetration</td>
</tr>
</tbody>
</table>

Modelling Architectural Design Information by Features
These categories are used for the documentation of building elements, products, and materials at three levels:

level 1 Gives a short description and some principles of functional elements that are of importance to designers and principals in the first phases of planning.

level 2 Provides a subdivision of the level 1 functional elements in product groups, including more detailed information so to allow a comparison of their technical properties and criteria for making the right choices.

level 3 Contains detailed technical product and material information of products available on the Dutch market in each of the product groups as classified by the NL-SfB method. The information is given in a standard format and can therefore easily be compared between similar products.

The first two levels have been developed by NBD in conjunction with other authorities in the Dutch Building & Construction industry, while the third level, the most extensive one, is based on information provided by the participating manufacturers.

6.4 Architectural information revealed in product models

This section analyses what type of information is generally laid out in the results of product modelling research. From the product models that have been reviewed during the course of this project, a selection of two product models is discussed here, the first of which is the conceptual model of spaces, space boundaries, and enclosing structures by Björk [1992]: This model is based on an analysis of four selected product models (i.e. RATAS by VTT, Finland; Synthesis Model by GSD, France; IDM from the EC-project COMBINE; and the House Model by [De Waard 1992]) and can be regarded as a
synthesis model that represents the most relevant, common concepts in these models. A summary of the major aspects in this model is listed in table 6.5. The complete model by Björk is shown in figure 6.4 on page 105.

Table 6.5 Analysed source: Björk’s synthesis model of spaces, space boundaries, and enclosing structures.

<table>
<thead>
<tr>
<th>Major aspects in Björk’s model of spaces, space boundaries, and enclosing structures (from [Björk 1992])</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spaces, subspaces</td>
<td>“A space is a volume bounded on all sides by enclosing structures which form the physical space boundaries of the space.”</td>
</tr>
<tr>
<td>space boundaries (both physical and imaginary)</td>
<td>A space boundary is an abstract concept which represents a part of the infinitesimally thin skin which surrounds a space or enclosing structures that bound it. Subspaces can in addition to physical boundaries also have imaginary space boundaries.</td>
</tr>
<tr>
<td>enclosing structures</td>
<td>An enclosing structure is an aggregation of objects which form the space boundaries of two or more individual spaces. […] In design, enclosing structures are often the basic unit with which enclosures are defined. Only in later stages of design they need to be broken down into smaller units.</td>
</tr>
<tr>
<td>decomposition of structures into components</td>
<td>A component is a clearly delimited part of an enclosing structure, which often is prefabricated and fastened to other components on site using joints or seams. In some cases the same component can be part of several enclosing structures at the same time.</td>
</tr>
<tr>
<td>distinction of components into static and opening components</td>
<td>An opening component is an abstract generalisation of doors and windows. A static component is an enclosing structure component which is immovable. Static components have at least a surface layer, many have a multi-layered structure.</td>
</tr>
<tr>
<td>decomposition of components into layers</td>
<td>A layer is a continuous volume of uniform material inside or on the surface of an enclosing structure component. An internal layer is invisible and its aesthetic outlook has no relevance. A surface layer is visible and consists of one to many surfaces.</td>
</tr>
<tr>
<td>surfaces and surface finishes of layers</td>
<td>A surface is an area of the outermost layer of an enclosing structure, which is uniform in material, colour, and surface treatment. A physical space boundary can be formed by one or many surfaces. A surface finish collects information about the material, colour, and surface treatment of a uniform surface.</td>
</tr>
<tr>
<td>assemblies of spaces, assemblies of space boundaries, assemblies of enclosing structures</td>
<td>A space assembly is an abstract, generic concept, which can be further sub-typed into other entities useful for information management. Consists of one to many spaces. Examples are: storey, fire zone, apartment.”</td>
</tr>
</tbody>
</table>

The level of detail in this model is not very high, however, the model can be called representative for the types of information structures and relationships that are generally found in product models in B&C research. The point of departure for the model has been to look at a building as a network of spaces separated by building elements,
therefore there is a strong emphasis on the relationships between spaces, space boundaries, and enclosing structures. Not included in the model are entities for the building as a whole, for building systems and sub-systems (the more aggregated entities in product models), and any non-enclosing structures that can be part of these building systems (e.g. equipment, installations, facilities, etc.). Also, though not explicitly stated so by Björk, the model shows a viewpoint that would generally represent the position of an architectural designer in an interdisciplinary product modelling environment. Many aspects from other disciplines in construction, such as structural engineering and HVAC engineering, are not addressed.

A much higher level of detail is reached in the second reviewed model, the Integrated Data Model (IDM+) developed in the EC-funded COMBINE projects [Augenbroe 1993, 1995]. This model, resulting from the EC programme JOULE on energy saving technology, is focused on energy and HVAC disciplines. However, the core of this model addresses information of general purpose in the B&C industry. The information in the tables 6.6 to 6.9 shows a selection of entities and relationships that are found in the IDM+ schemata. The entities must be interpreted as the type of objects and attributes of objects that are identified in the design domain for which the schemata are defined. A graphical representation of part of the Core schema of the IDM+ is shown in figure 6-5 on page 106.

The relations explicitly name the roles of the attributes of an object in the context of the object. In many cases, the relationship indicates this role using a verb or a verb in conjunction with a noun: e.g. the relation between the entity 'materialized layer' and the entity 'material' is 'has material'. In many other cases, however, the role is indicated by a term that specifies an explicit context for the related entity: e.g. the entity 'direction' has a relation to the entity 'vector' with the role 'orientation'; the entity 'length measure' relates to the same entity 'vector' with the role 'magnitude'. Therefore, the relations as listed in the tables below are to be regarded as 'entities' of information, just as the entities in the tables are.

The IDM+ was completed at the end of the year 1995. The complete models are provided in the EXPRESS language and NIAM² diagrams and are available at http://duce15.tudelft.nl/~combine.

2 Nijssen's Information Analysis Method.

3 This URL has been checked for validity at the time of publication of this thesis.
Table 6.6 Analysed source: Integrated Data Model from the European COMBINE project; entities and relations found in the Core schema.

<table>
<thead>
<tr>
<th>Entities and relations from IDM schema Core</th>
<th>Relations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>entities</strong></td>
<td><strong>relations</strong></td>
</tr>
<tr>
<td>object</td>
<td>composed of</td>
</tr>
<tr>
<td>performance</td>
<td>adjacent to</td>
</tr>
<tr>
<td>address</td>
<td>includes</td>
</tr>
<tr>
<td>physical object</td>
<td>bounded by</td>
</tr>
<tr>
<td>representation item</td>
<td>represented by</td>
</tr>
<tr>
<td>measure value</td>
<td>direction of</td>
</tr>
<tr>
<td>(length, mass, time, volume, temperature, etc.)</td>
<td>offset of</td>
</tr>
<tr>
<td>space object</td>
<td>of type</td>
</tr>
<tr>
<td>ground space</td>
<td>has ...</td>
</tr>
<tr>
<td>building space</td>
<td>has material</td>
</tr>
<tr>
<td>elementary space</td>
<td>has treatment</td>
</tr>
<tr>
<td>zone</td>
<td>has finish</td>
</tr>
<tr>
<td>internal space</td>
<td>has id</td>
</tr>
<tr>
<td>storey</td>
<td>has frame</td>
</tr>
<tr>
<td>building element</td>
<td>has opening method</td>
</tr>
<tr>
<td>catalog</td>
<td>has ... type</td>
</tr>
<tr>
<td>enclosing element</td>
<td>thickness</td>
</tr>
<tr>
<td>wall</td>
<td>connected to</td>
</tr>
<tr>
<td>ceiling</td>
<td>orientation</td>
</tr>
<tr>
<td>floor</td>
<td>ends</td>
</tr>
<tr>
<td>window</td>
<td>geometry of</td>
</tr>
<tr>
<td>enclosing element segment</td>
<td>location</td>
</tr>
<tr>
<td>hole</td>
<td>position</td>
</tr>
<tr>
<td>joint</td>
<td>points</td>
</tr>
<tr>
<td>finish</td>
<td>coordinates</td>
</tr>
<tr>
<td>finish surface treatment</td>
<td>sound resistivity index</td>
</tr>
<tr>
<td>(polished, unpolished smooth, rough)</td>
<td>u value</td>
</tr>
<tr>
<td>construction type</td>
<td>young's modulus</td>
</tr>
<tr>
<td>layer</td>
<td>elasticity</td>
</tr>
<tr>
<td>materialized layer</td>
<td>viscosity</td>
</tr>
<tr>
<td>representation</td>
<td>thickness</td>
</tr>
<tr>
<td>face</td>
<td>extinction coefficient</td>
</tr>
<tr>
<td>loop</td>
<td>refractive index</td>
</tr>
<tr>
<td>edge</td>
<td></td>
</tr>
</tbody>
</table>

Modelling Architectural Design Information by Features
Table 6.7 Analysed source: Integrated Data Model from the European COMBINE project; entities and relations found in the Space Function schema.

<table>
<thead>
<tr>
<th>Entities and relations from IDM schema</th>
<th>Space Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>entities</strong></td>
<td><strong>relations</strong></td>
</tr>
<tr>
<td>space function</td>
<td>max people number</td>
</tr>
<tr>
<td>... space function</td>
<td>used for</td>
</tr>
<tr>
<td>... space use</td>
<td>specifies</td>
</tr>
<tr>
<td>(for building space, elementary space, etc.)</td>
<td>composed of predicted level</td>
</tr>
<tr>
<td>zone function</td>
<td>design heating/cooling setpoint temperature</td>
</tr>
<tr>
<td>zone use</td>
<td>required ... quality</td>
</tr>
<tr>
<td>space usage</td>
<td>required ... permissible ...</td>
</tr>
<tr>
<td>temperature setting</td>
<td>... contribution</td>
</tr>
<tr>
<td>indoor air quality requirement</td>
<td>... gain</td>
</tr>
<tr>
<td>contaminant</td>
<td>number of ...</td>
</tr>
<tr>
<td>contaminant level</td>
<td></td>
</tr>
<tr>
<td>contaminant level unit</td>
<td></td>
</tr>
<tr>
<td>(one of: micrograms/m3, milligrams/m3, nCi/m3, fibre*10^6/m3, %vol)</td>
<td></td>
</tr>
<tr>
<td>pollutant contribution</td>
<td></td>
</tr>
<tr>
<td>lighting requirement</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.8 Analysed source: Integrated Data Model from the European COMBINE project; entities and relations found in the ESP Performance schema.

<table>
<thead>
<tr>
<th>Entities and relations from IDM schema</th>
<th>ESP Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>entities</strong></td>
<td><strong>relations</strong></td>
</tr>
<tr>
<td>performance</td>
<td>attribute names similar to entities</td>
</tr>
<tr>
<td>zone performance</td>
<td></td>
</tr>
<tr>
<td>lighting performance</td>
<td></td>
</tr>
<tr>
<td>sky conditions</td>
<td></td>
</tr>
<tr>
<td>(clear or overcast)</td>
<td></td>
</tr>
<tr>
<td>daylight factor</td>
<td></td>
</tr>
<tr>
<td>thermal performance</td>
<td></td>
</tr>
<tr>
<td>season</td>
<td></td>
</tr>
<tr>
<td>(summer or winter)</td>
<td></td>
</tr>
<tr>
<td>thermal comfort</td>
<td></td>
</tr>
<tr>
<td>relative humidity (%)</td>
<td></td>
</tr>
<tr>
<td>temperature (deg C)</td>
<td></td>
</tr>
<tr>
<td>heating</td>
<td></td>
</tr>
<tr>
<td>cooling</td>
<td></td>
</tr>
<tr>
<td>thermal power (W)</td>
<td></td>
</tr>
</tbody>
</table>

Modelling Architectural Design Information by Features
Table 6.9 Analysed source: Integrated Data Model from the European COMBINE project; entities and relations found in diagram 3 of the Technical System schema.

<table>
<thead>
<tr>
<th>Entities and relations from IDM schema Technical System (diagram 3: pipes, ducts, and wires)</th>
<th>relations</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipe length measure duct hvac shape rectangular round</td>
<td>flow performance velocity pressure fluid flow rate wire rated current</td>
</tr>
</tbody>
</table>

Product models and classifications can hardly be compared in any sense. However, the level of semantics in product models such as the IDM is, compared to that in classifications, much more detailed. The kind of entities included in these models reach a greater detail in the description of a building. Moreover, the relationships between entities, e.g. between building parts and their properties, are declared in detail. This is possible, because product models generally serve a limited purpose, a range of applications.

The analysis of product models, as shown in the review above, has therefore provided a significantly detailed overview, albeit within the domain for which these models were developed. In the next section, this analysis, together with the analysis of the other sources of architectural information, is used in a case study on a more narrow part of the architectural domain.

6.5 A case study on an inventory of architectural information

This case study aims at getting more insight in the information and type of information that is related to a specific type of concept in architectural design: vertical space boundaries. The concept of vertical space boundaries is chosen because sufficient information related to this concept is available in all of the selected sources, which attributes to the relevancy of the analysis. The inventory that results from this small case study illustrates the complexity and extent of information related to even the most trivial type of design concepts. Together with the analysis of architectural information sources it forms the starting point for defining classes of Features in architecture (see figure 6-1). The three groups of sources as discussed in the previous sub-sections are analysed within the scope of vertical space boundaries. This analysis leads to a rough overview of all information related to vertical space boundaries in design and construction contexts. Naturally, the various sources have to be regarded within the context of their development, however, the resulting inventory will provide a satisfactory basis for further development of classes of information in the Feature-based approach.

For some of the analysed sources not all the categories found in relation to vertical space boundaries are presented here, just those related to interior, vertical space boundaries, or even a subset of these. For the purpose of this case study, this restriction...
has not been made, but for reasons of brevity the complete results are not included here. This should, however, not affect the reasoning based on the result of the case study and the development into a categorisation of Features in the next section.

**Classifications of information related to vertical space boundaries**

Two classifications have been analysed on the subject of vertical space boundaries. First, table 6.10 lists the entries in the NL-SfB table-1 [SBK 1991] that relate to *interior, vertical space boundaries*. A brief description of the elements, their function, and some examples of elements included in these categories are given.

<table>
<thead>
<tr>
<th>code</th>
<th>contained elements in the NL-SfB classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td><strong>interior walls</strong></td>
</tr>
<tr>
<td>.1</td>
<td>non-structural description:</td>
</tr>
<tr>
<td></td>
<td>non-structural interior walls, forming the boundaries of the building, considered from top of the underlying floor, to the bottom of the above floor or roof.</td>
</tr>
<tr>
<td></td>
<td>function: separation of spaces (acoustics, security, climate, vision).</td>
</tr>
<tr>
<td></td>
<td>includes: insulation-facilities, finishes that are part of the wall, anchors, connections, dilatation-joints.</td>
</tr>
<tr>
<td>.10</td>
<td>general</td>
</tr>
<tr>
<td>.11</td>
<td>solid walls</td>
</tr>
<tr>
<td>.12</td>
<td>cavity walls</td>
</tr>
<tr>
<td>.13</td>
<td>fixed prefabricated wall units</td>
</tr>
<tr>
<td>.14</td>
<td>displaceable prefabricated wall units</td>
</tr>
<tr>
<td>.2</td>
<td>structural</td>
</tr>
<tr>
<td></td>
<td>description: structural interior walls, forming the boundaries of the building, considered from top of the foundations, until the top of the roof-constructions.</td>
</tr>
<tr>
<td></td>
<td>function: bearing structure of the building.</td>
</tr>
<tr>
<td></td>
<td>includes: separation of spaces (acoustics, security, climate, vision).</td>
</tr>
<tr>
<td></td>
<td>insulation-facilities, finishes that are part of the wall, anchors, connections, dilatation-joints.</td>
</tr>
<tr>
<td>.20</td>
<td>general</td>
</tr>
<tr>
<td>.21</td>
<td>solid walls</td>
</tr>
<tr>
<td>.22</td>
<td>cavity walls</td>
</tr>
<tr>
<td>.23</td>
<td>fixed prefabricated wall units</td>
</tr>
<tr>
<td>32</td>
<td><strong>openings in interior walls</strong></td>
</tr>
<tr>
<td>.1</td>
<td>not filled in</td>
</tr>
<tr>
<td></td>
<td>description: not filled in openings in interior walls.</td>
</tr>
<tr>
<td></td>
<td>function: passageway for traffic and vision between inner-spaces.</td>
</tr>
<tr>
<td></td>
<td>includes: openings for installations, connections, sills, finishes.</td>
</tr>
<tr>
<td>.10</td>
<td>general</td>
</tr>
<tr>
<td>.11</td>
<td>passageway for traffic</td>
</tr>
<tr>
<td>.12</td>
<td>passageway for vision</td>
</tr>
<tr>
<td>code</td>
<td>contained elements in the NL-SfB classification</td>
</tr>
<tr>
<td>------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>.2</td>
<td>filled with windows</td>
</tr>
<tr>
<td></td>
<td>description: openings in interior walls filled with window-frames and windows.</td>
</tr>
<tr>
<td></td>
<td>function: passageway for vision between inner-spaces</td>
</tr>
<tr>
<td></td>
<td>separation of inner-spaces (acoustics, security, climate, vision).</td>
</tr>
<tr>
<td></td>
<td>includes: connections, hinges, locks, sills, operation-facilities, glazing, finishes, security-facilities.</td>
</tr>
<tr>
<td>.20</td>
<td>general</td>
</tr>
<tr>
<td>.21</td>
<td>closed frames</td>
</tr>
<tr>
<td>.22</td>
<td>casement windows</td>
</tr>
<tr>
<td>.23</td>
<td>sash windows</td>
</tr>
<tr>
<td>.24</td>
<td>revolving windows</td>
</tr>
<tr>
<td>.25</td>
<td>combination windows</td>
</tr>
<tr>
<td>.3</td>
<td>filled with doors</td>
</tr>
<tr>
<td></td>
<td>description: openings in interior walls filled with doorframes and doors.</td>
</tr>
<tr>
<td></td>
<td>function: passageway for traffic and vision between inner-spaces</td>
</tr>
<tr>
<td></td>
<td>separation of inner-spaces (acoustics, security, climate, vision).</td>
</tr>
<tr>
<td></td>
<td>includes: shutters, connections, hinges, locks, sills, operation-facilities, glazing, finishes, security-facilities.</td>
</tr>
<tr>
<td>.30</td>
<td>general</td>
</tr>
<tr>
<td>.31</td>
<td>hinged doors</td>
</tr>
<tr>
<td>.32</td>
<td>sliding doors</td>
</tr>
<tr>
<td>.33</td>
<td>overhead doors</td>
</tr>
<tr>
<td>.34</td>
<td>revolving doors</td>
</tr>
<tr>
<td>.4</td>
<td>filled with facade-elements</td>
</tr>
<tr>
<td></td>
<td>description: openings in interior walls filled with facade-elements including windows and doors.</td>
</tr>
<tr>
<td></td>
<td>function: passageway for traffic and vision between inner-spaces</td>
</tr>
<tr>
<td></td>
<td>separation of inner-spaces (acoustics, security, climate, vision).</td>
</tr>
<tr>
<td></td>
<td>includes: connections, hinges, locks, sills, awnings, operation-facilities, glazing, finishes, security-facilities.</td>
</tr>
<tr>
<td>.40</td>
<td>general</td>
</tr>
<tr>
<td>.41</td>
<td>closed facade-elements</td>
</tr>
<tr>
<td>38.1</td>
<td>non-bearing interior building systems</td>
</tr>
<tr>
<td></td>
<td>description: complete building-systems for separation of inner-spaces and filling of openings in interior walls, not being part of the bearing structure of the building and not separable into the codes 31 to 37 (including floors, balustrades, and roofs).</td>
</tr>
<tr>
<td></td>
<td>function: passageway for traffic and vision between inner-spaces</td>
</tr>
<tr>
<td></td>
<td>separation of inner-spaces (acoustics, security, climate, vision).</td>
</tr>
<tr>
<td></td>
<td>includes: parts of the system forming walls, openings, installations, ceilings, non-structural floors, finishes.</td>
</tr>
</tbody>
</table>

The second classification analysed on the subject of vertical space boundaries is the set of proposed conversion tables in ICIS WG3, by Woestenenk [1995]. The entries in table 6.11 are the parts (recognisable, physical decompositions of construction types) and elements (possibly not yet physical parts within a construction type that has a particular function) in a building that relate to vertical space boundaries.
Table 6.11 Vertical space boundaries in the ICIS WG3 conversion tables.

<table>
<thead>
<tr>
<th>elements (functions) in ICIS WG3 table</th>
<th>parts</th>
</tr>
</thead>
<tbody>
<tr>
<td>physical function: structural: fixing</td>
<td>building, construction: wall, equipment; external space, construction: wall, anchor</td>
</tr>
<tr>
<td>physical function: structural: spanning</td>
<td>building, construction: wall, lintel; building, construction: beam</td>
</tr>
<tr>
<td>physical function: structural: support</td>
<td>building, construction: column; building, construction: wall</td>
</tr>
<tr>
<td>space function: protection: buffer</td>
<td>building, construction: wall, equipment; building, construction: wall: door, equipment</td>
</tr>
<tr>
<td>space function: protection: contamination</td>
<td>building, construction: wall, equipment</td>
</tr>
<tr>
<td>space function: space division</td>
<td>building, construction: balustrade; building, construction: partition; building, construction: partition: door; building, construction: partition: door/window; building, construction: partition: panel; building, construction: wall; building, construction: wall, equipment; building, construction: wall: door; building, construction: wall: door, equipment; building, construction: wall: door/window; building, construction: wall: gate; building, construction: wall: hatch; building, construction: wall: panel; building, construction: wall: portal; building, construction: wall: window; building, construction: wall: window, equipment; building, construction: wall: window, finish; building, equipment: screen; external space, construction: balustrade; external space, construction: enclosure; external space, construction: enclosure: gate; external space, construction: portal; external space, construction: wall; external space, construction: wall: door/window</td>
</tr>
</tbody>
</table>

Product- and material documentation on vertical space boundaries

From the construction products and materials documentation provided in the NBD [1997], the following categories of information related to interior walls can be retrieved.

At level 1 a description and the major principles of designing and constructing interior walls are given. Interior walls are divided in the following categories:

- in situ cast concrete wall
  reinforced concrete or light concrete, in situ cast
  application: construction walls
• prefabricated wall
delivered on site in its entirety
prefabricated components, reinforced concrete, or timber framework with
boarding.
application: construction walls
• element-wall
prefab wall units, height between floors
linked together, possibly with separate stiles
application: light separation walls
• assembly-wall
on site assembled from different materials
timber or metal framework with boarding and insulating filling
application: light separation walls, may be supporting
• brick wall
bricks and blocks for interior walls
joint with mortar or adhesive
application: dependent on the quality of the material

Level 1 provides directives for quality-assurance of the design and construction of
walls. The information is structured according to the main categories as described in
section 6.3 and is generic for the elements belonging to the group of interior walls. Some
aspects of the directives at this level are:

• surface finish and appearance
• seamless or partitioned elements
• regulations concerning strength and stability, including a description of tests and
allowable influences by changes in thermal conditions
• requirements for the joints and connections to other building elements
• material-properties and the physical consequences of construction-types and
design
• application:
  ▪ considerations concerning the function of the wall (constructive, separating)
  ▪ possibilities for installing electrical wiring and other piping
  ▪ possibilities for support of objects on the wall
  ▪ possible surface finishes
  ▪ walls as permanent or semi-permanent separation structure
• processing: directives for construction-planning and influences of processing on
the quality.

At level 2 detailed information is specified for each of the elements distinguished at level
1 (in situ cast, prefab, bricks, etc.). The information is structured in a similar manner as
at level 1, but contains more details specific for each class of elements. These details address aspects such as:

- structural principles
- joints of elements
- basic materials
- accessories
- generic characteristics of the elements (dimensions, weight, surface-structure, mechanical properties)
- design-aspects (structural aspects, physical problems, application aspects)

The information is specific for types of products belonging to one class of interior walls.

At level 3 detailed technical product and material information is provided by manufacturers. The list in table 6.12 is an indication of the kind of information that is found in the documentation on a subset of the products and materials for interior walls: concrete blocks and bricks for interior walls.

**Table 6.12 Vertical space boundaries in the NBD documentation level 3.**

<table>
<thead>
<tr>
<th>code</th>
<th>contained information in the NBD documentation level 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>short description</td>
</tr>
</tbody>
</table>
| E    | composition
|      | composition of the system:
|      | facing brickwork: one-sided (cavity wall), double-sided facing (interior wall)
|      | surface finish: structure, smooth, washed (only one-sided), or split-structure
|      | non-facing brickwork:
|      | surface finish: smooth
|      | blocks: smooth surface for finishing layer
|      | composition of the individual elements:
|      | massive bricks with structured, smooth, washed, or split-structure surface.
|      | material: concrete-composition: ...
|      | method of fabrication: ...
|      | accessories: conduit-bricks, notch-bricks, extra header-bricks, shaped bricks. |
| F    | shape, dimensions, weight
|      | shape: oblong
|      | dimensions and weight:
|      | length, width, height in mm
|      | weight in kg / piece
|      | consumption in number / m2 |
| G    | appearance
|      | surface-structure, colour:
|      | description of possible surface-finishes and colours |
| I    | mechanical properties
|      | material: strength:
<p>|      | compression-strength in MPa = N/mm² |</p>
<table>
<thead>
<tr>
<th>code</th>
<th>contained information in the NBD documentation level 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>fire, explosion</td>
</tr>
<tr>
<td></td>
<td>fire-resistance:</td>
</tr>
<tr>
<td></td>
<td>fire-resistance in minutes</td>
</tr>
<tr>
<td>M</td>
<td>thermal properties</td>
</tr>
<tr>
<td></td>
<td>transmission:</td>
</tr>
<tr>
<td></td>
<td>warmth-transmission-coefficient in $W/(m \cdot K)$</td>
</tr>
<tr>
<td>P</td>
<td>acoustic properties</td>
</tr>
<tr>
<td></td>
<td>air-sound-insulation:</td>
</tr>
<tr>
<td></td>
<td>minimum performance in dB</td>
</tr>
<tr>
<td></td>
<td>sound-absorption:</td>
</tr>
<tr>
<td></td>
<td>in % at different frequencies</td>
</tr>
<tr>
<td>R</td>
<td>energy and other factors</td>
</tr>
<tr>
<td></td>
<td>material-consumption:</td>
</tr>
<tr>
<td></td>
<td>see dimensions and weight under F.</td>
</tr>
<tr>
<td>T</td>
<td>applicability, design</td>
</tr>
<tr>
<td></td>
<td>functional usability:</td>
</tr>
<tr>
<td>V</td>
<td>processing characteristics</td>
</tr>
<tr>
<td></td>
<td>transport</td>
</tr>
<tr>
<td></td>
<td>storage</td>
</tr>
<tr>
<td></td>
<td>preparation</td>
</tr>
<tr>
<td></td>
<td>application</td>
</tr>
<tr>
<td></td>
<td>finishing</td>
</tr>
<tr>
<td>Y</td>
<td>economical, commercial factors</td>
</tr>
<tr>
<td></td>
<td>prices</td>
</tr>
<tr>
<td></td>
<td>delivery: conditions, time, district</td>
</tr>
<tr>
<td></td>
<td>guarantee</td>
</tr>
<tr>
<td></td>
<td>technical service</td>
</tr>
<tr>
<td>Z</td>
<td>references</td>
</tr>
</tbody>
</table>

**Vertical space boundaries in product models**

The synthesis product model by Björk, as reviewed in section 6.4, contains elements that are nearly all related to space boundaries. Figure 6-4 shows the model in the EXPRESS-G notation, its major entities are explained in table 6.5 on page 94. One of the main aspects in this model that are of relevance to the notion of space boundaries, is the distinction between physical and imaginary space boundaries. Although imaginary space boundaries were not further defined, the recognition of their existence in a product model was novel at the time of development of this model. However, the exact relationship between physical space boundaries and the enclosing entities that embody these boundaries in the physical building remains point of discussion. In conceptual design, a space boundary is normally geometrically represented by some form of face without thickness. The enclosing entity normally has a solid geometry and thus has thickness. The geometric relationship between the two is ambiguous and mostly a...

---

*Modelling Architectural Design Information by Features*
matter of choice. Two trivial options for this relationship are shown in figure 6-3. With the second option, the space boundary is situated in the centre of the enclosing entity; it is shared by the two spaces on either side of the enclosure. The other option assigns the space boundary to the inside of the enclosing entity, as seen from the bounded space; the space boundary in this case is not shared by two spaces. The model by Björk does not make a choice but rather allows both ways of modelling.

Other notable concepts in the model by Björk are the notions of aggregation and decomposition which are both present in the model. Spaces can be aggregated into space assemblies, but at the same time a space may have subspaces. Similarly, an enclosing entity can be aggregated into an enclosing entity assembly, while at the same time it can be decomposed, in the form of an enclosing structure, into components or sections. Finally, the structure of the entities layer and surface form an additional relationship between components and space boundaries.
The COMBINE IDM+ model exhibits much more detail. Table 6.6 on page 96 shows all the entities and relations found in the Core schema of this model, some of which are graphically represented in figure 6-5. This diagram shows an interpretation of two of the original NIAM diagrams from the IDM+. It includes those entities that are directly related to the configuration of spaces and enclosures in the model. Some notable aspects in this diagram are discussed, partly in relation with the previously discussed model by Björg.

![Diagram](image-url)
The IDM+ model shows a decomposition of spaces, as does the model by Björk, but here a space is differentiated into several subtypes of spaces, including building space, space subsystem, and elementary space. The modelling of information concerning the building as a whole is thereby made explicit. The notion of aggregation of spaces also returns in this model, by way of the entity ‘zone’ and its derived entity ‘storey’. Space boundaries in the IDM+ model do not exist as such. The entity in this model that comes closest to the notion of a space boundary, as defined by Björk, is the enclosing element which forms a bridge between spaces and materials.

The differentiation into various kinds of spaces allows the definition of different relationships of these spaces to their corresponding enclosures. This, in turn, allows modelling a decomposition of enclosures. A similar structure is present in the model by Björk, albeit less explicit. Another kind of decomposition of enclosing elements is present in the IDM+ model, namely through the inclusion of the entity ‘enclosing element segment’. This allows segmentation of any of the enclosing elements, and provides the way to model relationships with holes in the enclosures. Holes, in turn, are filled by enclosing elements again, a window for instance.

Adjacency of spaces is incorporated in the model, however only at the level of elementary spaces. Apparently the applications for which the model has been developed deal only with space to space relationships at that level.

The materialisation of enclosing elements is defined only at the lowest level of the decomposition of these elements, namely for elementary space enclosing elements. These elements, which can be walls, floors, etc., are related to a construction type. A construction type in turn is composed of layers that have a thickness and material. The geometric relation between the enclosing element and the construction type is determined by an offset and therefore made explicit, as opposed to the model by Björk.

Besides the direct relationship between spaces and enclosures, there also is an indirect relationship in the IDM+ model through the entity ‘finish’. This entity is an attribute of the entity ‘space’ and of the entity ‘enclosing element segment’. Although it cannot be read from the model whether or not the finish of a space is the same as the finish of the enclosing element segments that bound that space, it seems logical to assume that this consistency is maintained.

Not shown in the diagram of figure 6-5 is the definition of the entity ‘physical object’. In the IDM+, the physical object takes a very central position. Almost all other entities are directly or indirectly derived from this entity, some are derived from its parent, the top-level entity ‘idm object’. The physical object defines four recursive, generic relationships: ‘composed of’, ‘adjacent to’, ‘includes’, and ‘bounded by’. Since all derived entities inherit these relationships, virtually any entity instantiated from the model can specify these relationships to any other entity. Furthermore, the physical object defines a relationship to a collection of representations, topologic or geometric. Geometry in the COMBINE project is defined as a so-called Relational Model representation, which is considered a richer geometric representation than the often used Boundary representation.
6.6 Conclusions from the inventory

The sources described above concentrate on the communication of information of a certain type. In the case of the Elementenmethode '91 by SBK, the type of information is at a rather high abstraction level of physical elements in a building. The information on these elements can apply to design stages, or to stages of construction, management or maintenance. The elements are divided in generic classes, providing a typology of functional solutions to design problems, such as exterior wall, non-structural, cavity wall. It does not provide any properties or attributes of these elements, but merely concentrates on classifying the elements so that they can be uniquely identified and coded. For the definition of an information model, this classification can help build up the generic typology of information based on building elements, it does not help the development of the more detailed levels of information comprising the full description of these elements and their properties. Also it cannot be used for the development of an information model representing concepts and characteristics in architecture that are not directly related to physical building elements.

The classifications included and inter-related in the ICIS WG3 conversion tables by Woestenenk provide a taxonomy of functional solutions to design problems. As such, these tables are a useful source to inventory design concepts, though this inventory will be limited to those concepts that relate directly to solutions that can be attributed to physical elements in a building. More abstract concepts of design cannot be extracted from these classifications.

The NBD building documentation is intended to provide background information supporting design decisions. This information is generic at the first two levels and product-specific at the third level. The third level is the most extensive level. The information at the first two levels, although generic for building elements, isolates the elements in their technical context. The elements are not placed in the context of the overall building design and in the relations to for instance spaces, building systems, and other building elements. In other words, the information given does not address project specific aspects. Also abstract levels of building elements, where information is not yet known in much detail, are not addressed here. This source of documentation is useful in the development of information models in the sense that it provides an overview of the properties of building products and materials, therefore complementing at this physical level the element classifications discussed above. Although the documentation is always related to products, in this study it can be used to isolate from these products the information related to their properties, e.g. information on thermal properties.

The performance specifications in national building codes and regulations, which were not included in the case study, may form a basis for communicating performance requirements. However, there is no attempt to formalise the contents of the information being communicated. Performance specifications concern the quantitative and qualitative levels of requirements and do not provide a framework within which the requirements can be fitted or from which standard requirements can be taken as a basis for requirement-specification. The requirements need to be acquired from the complex set
of regulations or from existing standards for requirement specification. The existing methods for requirement specification serve as the textual counterpart of the set of technical drawings in a traditional, finalised design project and do not support the development of information models in a direct manner, since the information embedded in them is not structured in an orderly manner. Obviously they do form a valuable source for obtaining a detailed overview of the information need from a requirements point of view, in the domain for which information is modelled.

Of the reviewed product models, the Space model by Björk serves the very general purpose of finding common ground in a selection of previously developed product models. The focus is on the distinction in two ways of looking at a building design: from the spatial configuration point of view and from the physical elements point of view. The relation between these two points of view have been made explicit in this model. The IDM model has a much more explicit objective, given by the range of applications in the HVAC and energy discipline that are to use this model in a collaborative design context. The total IDM model consists of a number of schemata that represent the individual information need from the various computer applications, of which the common basis is collected in schemata like the core and space function schemata.

The investigated sources in this chapter provide an overview of what kind of information is relevant in building design, from early, conceptual design stages to stages of detailed, technical design of a building's equipment. The sources provide information concerning:

- elements and building parts;
- element and part properties;
- spaces;
- functions;
- products and materials;
- product and material properties;
- relationships between all of the above.

From this overview an initial inventory of concepts and characteristics is induced in the next section, leading to the development of a categorisation of Feature types in architecture.

### 6.7 Categories of Feature types in architecture

The survey in the case study in section 6.5 provides an overview of the kind of information that is related to vertical space boundaries. This overview is categorised in this section into a generic list of aspects related to vertical space boundaries. This list forms a collection of nominees for formalisation into Feature types, offering an insight in the kind of Feature types that can be expected to emerge in relation with vertical space boundaries. With the survey from the beginning of this chapter, this limited
categorisation is subsequently used to formulate a proposal for the general categorisation of Feature types related to the architectural discipline.

The objective of the continuation of the case study can thus be described as: to investigate the kind of information that is related to architectural concepts, such as space boundaries, that are likely to appear in future definitions as Feature types and in classifications of Feature types.

6.7.1 The case study continued

An analysis and interpretation of the results of the survey of sources of architectural information on vertical space boundaries as discussed in section 6.5 leads to the aspects listed in table 6.13. Examination of these aspects gives an indication to what type of Features are needed to represent this information. The conclusions following these considerations are displayed in the column on the right in the same table.

Table 6.13 Summary of aspects of vertical space boundaries: nominees for formalisation into Feature types.

<table>
<thead>
<tr>
<th>Design and planning aspects of vertical space boundaries</th>
<th>Discussion and conclusions towards required Feature types</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. spaces adjacent to vertical space boundary:</td>
<td>The aspect of adjacency plays a role in relation with both spaces and space-boundaries, forming an intermediary between the two. Therefore it seems relevant to model this aspect as a separate entity of information, thus defining a Feature type for it.</td>
</tr>
<tr>
<td>- only interior (i.e. we have an internal space boundary)</td>
<td>The property interior/exterior can be related to spaces and, although not relevant directly for the adjacency, it has an important influence on the space boundary adjacent to a space. The interior/exterior property has effects that extend the scope of the space, which is again a reason for modelling this information as an entity with its own relationships and behaviour: a Feature type.</td>
</tr>
<tr>
<td>- at least one exterior (i.e. we have an external space boundary)</td>
<td></td>
</tr>
<tr>
<td>2. function of vertical space boundary:</td>
<td>These are main functions of vertical space boundaries. The function of separating is further specified below, indicating a hierarchy of functions. ‘Function’ is a type of information that is expected to be relevant for almost every physical building element and for many non-physical concepts as well. A set of Feature types representing different types of functions will obviously be required.</td>
</tr>
<tr>
<td>- separating and part of structure</td>
<td></td>
</tr>
<tr>
<td>- separating and supporting</td>
<td></td>
</tr>
<tr>
<td>- only separating</td>
<td></td>
</tr>
<tr>
<td>3. type of separation function that needs to be performed:</td>
<td>The function of separation appears to be a very complex function that can be subdivided into many aspects. Each of these aspects has its own implications for the realisation of the function by physical building elements, and also for the surrounding parts of the building, such as the spaces that are separated. Clearly these functions can themselves be subdivided into many aspects, however, identification at this level seems</td>
</tr>
<tr>
<td>- visual separation</td>
<td></td>
</tr>
<tr>
<td>- acoustic separation</td>
<td></td>
</tr>
<tr>
<td>- organisational separation</td>
<td></td>
</tr>
<tr>
<td>- separation for reasons of security or health, e.g.:</td>
<td></td>
</tr>
<tr>
<td>- firewall</td>
<td></td>
</tr>
</tbody>
</table>

Modelling Architectural Design Information by Features.
### Categories of Feature types in architecture

#### Design and planning aspects of vertical space boundaries

<table>
<thead>
<tr>
<th>Design and planning aspects of vertical space boundaries</th>
<th>Discussion and conclusions towards required Feature types</th>
</tr>
</thead>
<tbody>
<tr>
<td>- airtight separation</td>
<td>appropriate.</td>
</tr>
<tr>
<td>- watertight separation</td>
<td>'Morphology', like 'function', can also be regarded as a</td>
</tr>
<tr>
<td>- other specific purpose separation, e.g. light seal</td>
<td>very general type of information, relevant to almost all</td>
</tr>
<tr>
<td></td>
<td>elements in a building-design and even to many non-</td>
</tr>
<tr>
<td></td>
<td>physical concepts. Different aspects of morphology can</td>
</tr>
<tr>
<td></td>
<td>be distinguished which leads to a decomposition of a</td>
</tr>
<tr>
<td></td>
<td>generic 'morphology' Feature type into more detailed</td>
</tr>
<tr>
<td></td>
<td>Feature types for 'shape', 'dimensions', 'position', and</td>
</tr>
<tr>
<td></td>
<td>'orientation'.</td>
</tr>
<tr>
<td>4. morphology of vertical space boundary:</td>
<td>'Heavy' or 'light' are aesthetic characteristics of a</td>
</tr>
<tr>
<td>- shape</td>
<td>vertical space boundary and typically form an abstract</td>
</tr>
<tr>
<td>- dimensions</td>
<td>concept in design that is not dealt with in traditional</td>
</tr>
<tr>
<td>- position</td>
<td>approaches towards information modelling or data</td>
</tr>
<tr>
<td>- orientation</td>
<td>classification. In a Feature model supporting early</td>
</tr>
<tr>
<td></td>
<td>design stages, it seems appropriate to include these</td>
</tr>
<tr>
<td></td>
<td>abstract concepts as well. They may well apply also to</td>
</tr>
<tr>
<td></td>
<td>other kinds of elements in a building.</td>
</tr>
<tr>
<td>5. character of vertical space boundary towards each</td>
<td>The degree to which a vertical space boundary is open is</td>
</tr>
<tr>
<td>adjacent space:</td>
<td>relevant not only in relation to a space boundary, but</td>
</tr>
<tr>
<td>- heavy separation</td>
<td>for instance also in the context of circulation in a</td>
</tr>
<tr>
<td>- light separation</td>
<td>building. It also relates with, for instance, the</td>
</tr>
<tr>
<td></td>
<td>separation function of a space boundary. This aspect has</td>
</tr>
<tr>
<td></td>
<td>aesthetic elements as well as functional elements, which</td>
</tr>
<tr>
<td></td>
<td>gives a Feature type representing this aspect a dual</td>
</tr>
<tr>
<td></td>
<td>significance.</td>
</tr>
<tr>
<td>6. openness of vertical space boundary:</td>
<td>This aspect identifies the different types of compositions</td>
</tr>
<tr>
<td>- completely closed</td>
<td>that can be applied for the physical creation of the</td>
</tr>
<tr>
<td>- can be (partly) opened for passage</td>
<td>vertical space boundary. Although some characteristics of</td>
</tr>
<tr>
<td>- is (partly) open for passage</td>
<td>the composition of space boundaries may be common for all</td>
</tr>
<tr>
<td>- has visual openings</td>
<td>of them, each of these types of compositions has its own</td>
</tr>
<tr>
<td></td>
<td>set of characteristics which distinguishes the</td>
</tr>
<tr>
<td></td>
<td>composition from the others: a layered composition has</td>
</tr>
<tr>
<td></td>
<td>types of characteristics that are different from those</td>
</tr>
<tr>
<td></td>
<td>describing masonry. A set of hierarchically structured</td>
</tr>
<tr>
<td></td>
<td>Feature types seems appropriate here.</td>
</tr>
</tbody>
</table>

#### Physical aspects of vertical space boundaries

<table>
<thead>
<tr>
<th>Physical aspects of vertical space boundaries</th>
<th>Discussion and conclusions towards required Feature types</th>
</tr>
</thead>
<tbody>
<tr>
<td>7. composition of vertical space boundary:</td>
<td>This aspect identifies the different types of compositions</td>
</tr>
<tr>
<td>- no physical composition, only an abstract boundary</td>
<td>that can be applied for the physical creation of the</td>
</tr>
<tr>
<td>of space (e.g. a line on the floor, a row of columns)</td>
<td>vertical space boundary. Although some characteristics of</td>
</tr>
<tr>
<td></td>
<td>the composition of space boundaries may be common for all</td>
</tr>
<tr>
<td></td>
<td>of them, each of these types of compositions has its own</td>
</tr>
<tr>
<td></td>
<td>set of characteristics which distinguishes the</td>
</tr>
<tr>
<td></td>
<td>composition from the others: a layered composition has</td>
</tr>
<tr>
<td></td>
<td>types of characteristics that are different from those</td>
</tr>
<tr>
<td></td>
<td>describing masonry. A set of hierarchically structured</td>
</tr>
<tr>
<td></td>
<td>Feature types seems appropriate here.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Modelling Architectural Design Information by Features*
<table>
<thead>
<tr>
<th>Physical aspects of vertical space boundaries</th>
<th>Discussion and conclusions towards required Feature types</th>
</tr>
</thead>
<tbody>
<tr>
<td>- hollow</td>
<td>A large range of physical performance aspects can be related to vertical space boundaries. Here, they are subdivided into those describing the performance of materials and those describing the performance of products. The latter ones are likely to depend on the performance of materials, but they also depend on the shape of the product and, for instance, its composition. Performance aspects, as those mentioned here, are relevant to a range of materials and products used for many purposes in construction. The position and role of these characteristics cannot be defined without knowing the specific context of the materials and products, however, their physical semantics has been well defined. The definition of Feature types for each of these characteristics would benefit the formalisation of the design domain. The definition of materials and products can simply define relationships to these Feature types as being the attributes that control their composition.</td>
</tr>
<tr>
<td>- mixture of the above</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>8. physical performance of vertical space boundary:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- material performance</td>
</tr>
<tr>
<td>- mechanical</td>
</tr>
<tr>
<td>(compression strength, elasticity, hardness, wear resistance)</td>
</tr>
<tr>
<td>- quality of surface</td>
</tr>
<tr>
<td>(durability, scratch resistance, optical properties)</td>
</tr>
<tr>
<td>- physical characteristics</td>
</tr>
<tr>
<td>(combustibility, water-tightness, flexibility)</td>
</tr>
<tr>
<td>- product performance</td>
</tr>
<tr>
<td>- permissible loads</td>
</tr>
<tr>
<td>- stability</td>
</tr>
<tr>
<td>- mechanical properties</td>
</tr>
<tr>
<td>(moment of inertia, dilatation, conductivity, absorption)</td>
</tr>
<tr>
<td>- physical characteristics</td>
</tr>
<tr>
<td>(sound-insulation, sound-absorption, sound-production)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>9. relationships to other building systems:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- functional relationships</td>
</tr>
<tr>
<td>(e.g. topological relationships to other walls, floors, piping)</td>
</tr>
<tr>
<td>- physical connections (e.g. joints with other walls, floors, windows)</td>
</tr>
</tbody>
</table>

Discussion and conclusions towards required Feature types

Functional relationships of vertical space boundaries are those relationships that enable the functionality of either the space boundary itself, or the related building systems. Often these functional relationships are embodied by physical connections, however, this is not necessarily so. In many case, these relationships are intermediary between the two related building systems, therefore they are best modelled independently as Feature types. One of the advantages this provides is that the relationships do not have to be limited to binary relationships, but may include multiple related systems, e.g. a wall related to two supported floors. Another is that, being a Feature by
6.7. A categorisation of Feature types

The considerations in conclusion of the case study, as presented in table 6.13, are generalised and lead to the proposition of a categorisation of Feature types for modelling architectural design information. Three main categories are distinguished:

1. Category of Conceptual Feature types

2. Category of Component Feature types

Conceptual Feature types and Component Feature types are the representatives for the main elements of an architectural design. The category of Conceptual Feature types contains those types that are considered the main entities that build up an architectural design. This includes such entities as spaces, living-units, floors, walls, HVAC systems, and so forth. This a very broad category that accommodates for abstract notions, such as spaces and space boundaries, and for physical elements like floors and walls, the physical embodiment of space boundaries. The category of Component Feature types is a subset of the category of Conceptual Feature types distinguishing all concepts that have a physical appearance in the building as eventually constructed, e.g. walls and floors, from
those concepts that will never have a physical presence in the building, e.g. spaces and imaginary space boundaries.

3. Category of Specification Feature types
   The third main category of Feature types consists of Specification Feature types. These are the kind of Feature types that specify the characteristics and behaviour of Conceptual Feature types. This category is further subdivided into the following categories:
   a. Function Feature types
   b. Form Feature types
   c. Physical Feature types
   d. Context Feature types
   e. Construction Feature types
   f. Life-cycle Feature types
   Each of these subcategories has its own subdivision, as is shown in figure 6-6.

The proposed categories of Feature types are graphically presented in figure 6-6. The inventory of Feature types in this chapter has resulted in an understanding of what kind of information is relevant when modelling an architectural design. This understanding

![Figure 6-6 Proposed categorisation of Feature types for architectural design information. The scheme uses the EXPRESS-G notation, he it that each entity-box represents a category of Feature types; a thin line indicates the general kind of relationship that exists between Feature types from one category to another; a thick line indicates the division of a category into subcategories.](image-url)
was necessary to develop an information modelling approach that aims at supporting the
dynamic nature of design, meeting the requirements of flexibility and extensibility. The
development and content of this framework is addressed in the next chapter. Also
addressed in the next chapter is the notion of Generic versus Specific Feature Types. In
this light, the categorisation of Feature types as shown in figure 6-6 is meant to be a
proposal for the taxonomy of standardised Generic Feature Types.
Part Three

Framework for Feature-Based Architectural Information Modelling

O essencial é saber ver.
Saber ver sem estar a pensar.
Saber ver quando se vê;
E nem pensar quando se vê
Nem ver quando se pensa.

Essentieel is kunnen zien,
Kunnen zien zonder te denken,
Kunnen zien wanneer men ziet,
En niet denken wanneer men ziet
Noch zien wanneer men denkt.

Fernando Pessoa, 1888 – 1935 (O que nós vemos das cousas são as cousas, Alberto Caeiro, 1914)
Vertaling: August Willemsen, 1991, p.95 (Wat wij zien van de dingen zijn de dingen)
This chapter introduces the framework for information modelling that is developed in this research project to fulfil the requirements of flexibility and extensibility posed by the dynamic nature of architectural design. It discusses the processes involved in defining and modelling information within this framework. The framework consists of an infrastructure of three abstraction layers that facilitates these processes. A schema for the processes of defining Feature Types and modelling Feature Instances is presented. The main part of the chapter is dedicated to the definition and discussion of the classes of Feature Types, the classes of Feature Instances, and their relationships. Graphical and textual notations are defined for both types and instances. The structures of libraries of Feature Types and models of Features are defined and the concept of incidental relationships, which are modelled only in actual design models, not in conceptual models, is introduced.
7.1 A framework for flexibility and extensibility

The theory presented thus far in this thesis addressed the requirements that must be met by the way design information models are defined and structured, and what type of information has to be dealt with in these models. This chapter develops a framework for information modelling that fulfils these requirements and can form the basis for the design and development of design support systems that incorporate the Feature-based modelling approach. Hence, the framework is called the Feature-Based Modelling framework or FBM framework.

The main issue in the development of such a framework is that the information models that result from working with this framework should meet the requirements of flexibility and extensibility that have been formulated in chapter 3. The development of the framework is based on object orientation, which is regarded the most appropriate technology to define and structure information within the context of this research project (see section 2.2.3 for an introduction to object orientation). A Feature therefore is an object in the framework, its definition and structure are provided in the object's type definition which is called the Feature Type. Accordingly, a design information model is built up from Features of which the definitions are formulated in conceptual models of Feature Types, also called Feature Type Libraries. The actual design information models, to distinguish them from conceptual models with Feature Types, are called instance models or Feature models.

Departing from the above, extensibility means that in the course of design, as more information becomes known and gets defined by a designer, Feature Types can be defined to meet the particular needs of the design context (see figure 7-1). These new Feature Types can then be added to the conceptual model of Feature Types. From these new Feature Types, instances can be created in the particular design information model, being the actual Features.

Flexibility (see figure 7-2) is required both at the level of the conceptual model and at the level of the instance model. For the extensibility to work, the structure of information defined in Feature Types in the conceptual model needs to be adjusted for new Feature Types to fit in. This is the flexibility at the

---

1 In fact, this is only one of two points of view from which instantiation takes place in the framework, see section 7.6 on this twofold instantiation.
conceptual level. It involves the addition and modification of relationships between Feature Types.

At the level of the instance model, which contains the actual Features, flexibility means that the definition of Features, and in particular their inter-relationships, allows restructuring the network of Features in accordance with the changing state of the design. This implies that relationships between Features are not rigidly defined and can be removed, added or modified at will by a designer. Which, in turn, implies that this kind of relationship is not defined at the conceptual level, but at the level of instantiated Features instead. This notion of relationships defined at the level of instantiated information models will play an important role in the development of the framework.

The following sections describe the information-modelling framework in terms of the processes that take place at the various levels within the framework, and the way information is structured within the framework.

7.2 Processes in Feature modelling

Essentially, four processes take place in the framework for Feature-based modelling, these are displayed in the IDEF-0 schema of figure 7-3. Although they are presented and discussed here in a successive order, they do not necessarily form a linear sequence, as is shown by the back looping data-flows. In Feature modelling activities, the described processes will take place in arbitrary order, as is required for the design-case at hand. The processes much resemble modelling activities in product modelling and other object-oriented approaches. Yet, apart from the resulting information structure which in the Feature-based approach differs largely from these approaches, the major difference in terms of processes is that the definition of the object type in the Feature-based approach also lies in the hands of the designer, and is not privileged to the developer of information models. This section introduces the four processes as shown in figure 7-3.

The first process to be discussed is called ‘Feature Type definition’ in which domain knowledge is transformed into a formal description: a Feature Type. A particular set of knowledge from the domain of design is identified as a relevant concept that will serve as a basic entity in modelling and reasoning about designs and that will be represented formally by a Feature Type. The Feature Type describes the kind of knowledge that is represented by this formalisation and how it is structured. It is later used to create instances of this concept in actual design models, where the formalised knowledge is applied in a particular design case. How domain knowledge can be formalised and what the resulting structures of information will look like, is discussed in more detail in sections 7.3 to 7.7. Various approaches that can be followed in defining Feature Types are discussed in section 8.1.

In the second process, Feature Types are classified into libraries of Feature Types. These libraries in fact form the conceptual models that represent domains of design-knowledge and are the basis for modelling information in actual design cases. Classification mainly serves two purposes. The first is to help a designer organise the formalised body of domain knowledge into categories of Feature Types. The second is to
allow standardised domain knowledge to be unambiguously accessible by those who conform to the standard and aim to use the standardised domain knowledge for modelling purposes and exchange of design information.

The third of the processes to be discussed is the one where Features are instantiated from Feature Types. Feature Instantiation involves the selection from a Feature Type Library of an appropriate Feature Type in the context of the current design case, and creating a Feature Instance based on the content-type and structure of information as defined in the Feature Type. The Feature Instance is then provided with the actual information as applicable for the particular design case, and is related to the structure of Feature Instances that are already present in the model representing the design case. Clearly, if no appropriate Feature Type is available for the specific circumstances in the current design, a new Feature Type must be defined prior to the instantiation procedure. How information is structured and how Features are interrelated is discussed in detail in sections 7.3 to 7.7, while section 7.8 discusses more elaborately the issues in instantiation of Feature Types. Section 8.3 presents a number of scenarios for Feature instantiation.

The fourth process taking place in Feature-based modelling and the final one to be discussed here, is modification of Feature models. Modification of the information represented in Feature Instances may take place in various forms, depending on the way
the information is structured in the Feature Instance and in the Feature Type, and depending on the effect of the modification as desired by the designer. The information present in the Feature Instance can simply be modified by changing, for instance, its numerical value. Another way of modifying the model is to change the inter-relationships between Feature Instances, which may result in different values for information in the model, or even in differently structured information. Before these various ways of modifying a Feature model are discussed further, the Feature-based structure of information is examined in the next three sections. Section 8.4 continues to address the issue of modifying Feature models.

7.3 The Feature-based information structure

The framework for Feature-based information modelling defines a three-layered information infrastructure (see figure 7-4). The three layers of abstraction in this model accommodate the required functionality of Feature models and conceptual Feature models (libraries of Feature Types) to enable extensibility and flexibility of these models. The dashed box in figure 7-4 indicates the middle layer containing Feature Type definitions. These are either Generic Feature Types or Specific Feature Types. The difference between the two and their relationship is explained in section 7.4; for now it suffices to describe Generic Feature Types as standardised and Specific Feature Types as customised, or designer-defined, Feature Types. The middle layer contains Feature Types collected in Feature Type Libraries. The Feature Types define the type of information that a Feature may contain and the structure of this information. For example, a Feature Type called 'Material' would have such properties as 'colour', 'texture', 'durability'. Each of these properties are defined as types of Features themselves and referred to by the 'Material' Feature Type. This ensures that property-definitions can be shared by various Feature Types and that duplicate definitions of such properties are not necessary.

The bottom layer of the three-layered model contains the actual information describing a particular design. This information is represented in collections of Feature Instances, also called Features in short. A collection of Feature Instances forms a Feature model. Feature Instances are instantiations of Feature Types, or conversely,
Feature Types define the structure and type of information contained in Feature Instances. An instance of the Feature Type 'Material' would be the Feature named 'Concrete', that has the value 'grey' for the property 'colour', the value 'rough' for the property 'texture', and the value 'high' for the property 'durability'. Each of these values of the properties are instances of the respective Feature Types and are related by reference to the Feature 'Concrete'. This structure is similar to the structure of property definitions in Feature Types, and is to ensure that duplicate instances of such properties are not necessary where usage of the same property is actually intended. The colour of another material that should match as closely as possible the colour of this concrete would simply refer to the same colour property. Using references for properties that are modelled as Features guarantees that such sharing of properties is always possible. Further discussion on the advantages and disadvantages of this modelling concept is found in section 8.4.

The top layer in the model in figure 7-4 is the so-called Meta Layer. This layer defines the format in which Feature Types and Feature Instances are to be defined. The format specifies the different kinds of Feature Types and Feature Instances that can be defined in the framework: these are classes of Feature Types and classes of Feature Instances. The remainder of this thesis will mainly focus on the definition and structure of these classes and their relationships to each other and design support systems. Classes of Feature Types are defined in section 7.7, classes of Feature Instances in section 7.8.

7.4 Generic versus Specific Feature Types

As indicated before, the difference between Generic Feature Types and Specific Feature Types is that Generic Feature Types are standardised, whereas Specific Feature Types are defined, for instance, by a designer for specific purposes. For Feature-based modelling systems, the difference between the Generic and Specific Feature Types has only limited relevance. Both categories can be extended with newly defined Feature Types, albeit that new Specific Feature Types will probably appear much more frequently than new Generic Feature Types. Feature Types can be defined 'from scratch', i.e. without an underlying Feature Type from which information is inherited, or as a specialisation of another Feature Type. Specialisation can occur within a category of Feature Types, but a Specific Feature Type may also be a specialisation of a Generic Feature Type.

The possible role of Generic Feature Types in data exchange issues is discussed in section 8.5 on standardisation and communication. This includes a comparison of this approach to the developments in STEP and the IFCs.
7.5 A typology of relationships

A survey of relationships between information entities in architectural product models has resulted in the distinction of the following kinds of relationships.

- specialisation relationship
- decomposition relationship
- structural dependencies
- adjacency
- tolerance
- dimensional relationship
- positional relationship
- existential dependency
- algorithmic relationship

These relationships are categorised into four types, three of which are known from object oriented approaches.

1. Specialisation
   Specialisation indicates that a given type of Feature is a sub-type of another type. The sub-type inherits all characteristics of the super-type and distinguishes itself from the super-type by adding characteristics: the result is a specialised type. Often this sort of relationship is denoted as an is_a relationship.

2. Decomposition
   Decomposition indicates another kind of hierarchy, in which information entities are divided into parts, or components. There is no inheritance, the components just contain a part of the total of characteristics. Decompositions are often called has_a relationships.

   The first two relationships, specialisation and decomposition, have been discussed also in section 2.2.3 which reviewed object-oriented abstraction mechanisms. The inverse mechanism of specialisation is called generalisation, while the inverse of decomposition is called aggregation.

3. Association
   Associations are not hierarchical relationships, there is no inheritance, nor division of characteristics. This relationship indicates any association of two entities of information that does not fall in either categories of specialisation or

---

2 The term 'entity' is used to refer to both Feature Types and Feature Instances. The reason for this is that the discussed relationships are applicable to both the conceptual and instantiated level of information modelling.
decomposition. However, a particular type of association is distinguished separately and called specification.

4. Specification
The above distinguished types of relationships appear in OO approaches to information analysis. A fourth type is distinguished in the FBM framework: a specification relationship is a kind of association indicating that one entity specifies information about another. A specification is used to model information that directly determines the characteristics of a concept. It is not to be confused with decomposition, since the specified information does not need to be a part of the concept. An example of a specification is a type of Feature called Door for which the manufacturer-details are specified by a type called Manufacturer. Note that the specification relationship is directed from the Door to the Manufacturer as ‘specified by’ and conversely from the Manufacturer to the Door as ‘specifies’.

Martin and Odell [1995] distinguish compositional relationships from non-compositional relationships. Compositional relationships are further distinguished based on the combination of three properties, concerning:

- Configuration: whether or not a functional or structural relationship exists between parts or between part and object.
- Homeomerity: when parts are the same kind of thing as the whole, they are homeomerous.
- Invariance: parts are invariant when they cannot be separated from the whole without destroying the whole.

Martin and Odell advance this into a classification of compositional relationships, as shown in table 7.1. Looking at these different kinds of compositions helps to understand the different types of relationships that are permitted in the FBM framework in this research project. The table lists how the kinds of compositions recognised by Martin and Odell are represented as Feature relationships in the framework of the project in this thesis.

Table 7.1 Compositional relationships [Martin and Odell 1995] and how they relate to the relationships in the FBM framework.

<table>
<thead>
<tr>
<th>compositional relationships (Martin and Odell [1995])</th>
<th>Feature relationships</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component - Integral Object composition</td>
<td>Decomposition</td>
</tr>
<tr>
<td>Defines a configuration of parts within a whole (Scenes - Film; Wheels - Cart).</td>
<td></td>
</tr>
<tr>
<td>Material - Object composition</td>
<td>Specification</td>
</tr>
<tr>
<td>Defines an invariant configuration of parts within a whole (Milk - Cappuccino; Iron - Car).</td>
<td></td>
</tr>
<tr>
<td>Compared to component - integral object, the parts in this case cannot be removed from the object.</td>
<td></td>
</tr>
</tbody>
</table>
A typology of relationships

<table>
<thead>
<tr>
<th>compositional relationships (Martin and Odell [1995])</th>
<th>Feature relationships</th>
</tr>
</thead>
<tbody>
<tr>
<td>Portion - Object composition</td>
<td>Decomposition</td>
</tr>
<tr>
<td>Defines a homeomorphic configuration of parts within a</td>
<td></td>
</tr>
<tr>
<td>whole (Metre - Kilometre; Slice - Loaf).</td>
<td></td>
</tr>
<tr>
<td>Compared to the first two, the parts are now of the</td>
<td></td>
</tr>
<tr>
<td>same kind as the object.</td>
<td></td>
</tr>
<tr>
<td>Place - Area composition</td>
<td>Decomposition</td>
</tr>
<tr>
<td>Defines a homeomorphic and invariant configuration of</td>
<td></td>
</tr>
<tr>
<td>parts within a whole (San Francisco - California; peak</td>
<td></td>
</tr>
<tr>
<td>- mountain).</td>
<td></td>
</tr>
<tr>
<td>Compared to the portion - object composition, here</td>
<td></td>
</tr>
<tr>
<td>each homeomorphic piece cannot be removed.</td>
<td></td>
</tr>
<tr>
<td>Member - Bunch composition</td>
<td>Decomposition</td>
</tr>
<tr>
<td>Defines a collection of parts as a whole (Tree -</td>
<td></td>
</tr>
<tr>
<td>Forest; Ship - Fleet).</td>
<td></td>
</tr>
<tr>
<td>In the composition relationships above, the parts</td>
<td></td>
</tr>
<tr>
<td>bear a particular functional or structural relationship</td>
<td></td>
</tr>
<tr>
<td>to one another or to the object they comprise. In</td>
<td></td>
</tr>
<tr>
<td>the member - bunch composition, membership of a</td>
<td></td>
</tr>
<tr>
<td>collection is the only requirement for a part to be</td>
<td></td>
</tr>
<tr>
<td>in the composition.</td>
<td></td>
</tr>
<tr>
<td>Member - Partnership composition</td>
<td>Decomposition</td>
</tr>
<tr>
<td>Defines an invariant collection of parts as a whole</td>
<td></td>
</tr>
<tr>
<td>(Stan Laurel - Laurel and Hardy).</td>
<td></td>
</tr>
<tr>
<td>As compared to member - bunch compositions, these</td>
<td></td>
</tr>
<tr>
<td>members cannot be removed from the whole.</td>
<td></td>
</tr>
</tbody>
</table>

All of the above kinds of compositions are modelled in the FBM framework as Decomposition relationships, except for the Material - Object relationship. In the perspective of this project, material characteristics are specifications of objects that are modelled with the Specification relationship. The other kinds of compositions are not further distinguished by the type of the relationship. However, the specific meaning of the relationship is to be expressed using the role name of the relationship.

Some examples of non-compositional kinds of relationships are given by Martin and Odell in contrast with the compositions. These are listed in table 7.2 and again compared to the Feature relationships.

Table 7.2 Non-compositional relationships [Martin and Odell 1995] and how they relate to the relationships in the FBM framework.

<table>
<thead>
<tr>
<th>non-compositional relationships (Martin and Odell [1995])</th>
<th>Feature relationships</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topological inclusion</td>
<td>Association</td>
</tr>
<tr>
<td>Examples: Customer in the Store; Meeting in the Afternoon.</td>
<td></td>
</tr>
<tr>
<td>Classification inclusion</td>
<td>Instantiation</td>
</tr>
<tr>
<td>Examples: Gone with the Wind is part of the set of objects of the class Book.</td>
<td></td>
</tr>
</tbody>
</table>
A Framework for Modelling Architectural Features

<table>
<thead>
<tr>
<th>non-compositional relationships (Martin and Odell [1995])</th>
<th>Feature relationships</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Attribution</strong></td>
<td><strong>Specification</strong></td>
</tr>
<tr>
<td>Examples: Lighthouse has Height and Weight, yet Height is not part of a Lighthouse.</td>
<td></td>
</tr>
<tr>
<td><strong>Attachment</strong></td>
<td><strong>Association</strong></td>
</tr>
<tr>
<td>Examples: Toes are attached to Feet, and are also part of Feet (= composition), yet Earrings are attached to Ears, but are not part of Ears.</td>
<td></td>
</tr>
<tr>
<td><strong>Ownership</strong></td>
<td><strong>Association</strong></td>
</tr>
<tr>
<td>Examples: A Bicycle has Wheels and Wheels are part of a Bicycle (= composition), yet a Girl has a Bicycle, which is not a part of the Girl.</td>
<td></td>
</tr>
</tbody>
</table>

The Classification inclusion is clearly a Instantiation in the FBM framework. From the other relationships, the attribution is modelled as a Specification relationship between Features, where, in the given example, the Height specifies a property of the Lighthouse. All other relationships are interpreted in this project as Associations between Features: the Meeting is associated with the Afternoon, the Bicycle is associated with the Girl.

An equivalent of the Specialisation or inheritance relationship is not mentioned in the classification by Martin and Odell.

**Application and interpretation of relationships**

Relationships in the framework can be distinguished at two levels. At the typological level, relationships are defined within Feature Types. A relationship may be optional, but in principle a relationship at typological level implies that all instances of the particular type will instantiate the relationship. In object oriented approaches, this level of relationship is defined using the first three categories shown above.

In addition, the framework also allows relationships between instances to be defined at the level of instances only, i.e. without being defined at the typological level. This new aspect in information modelling does not normally appear in OO approaches.

It is further discussed in section 7.9, after the introduction and discussion of the classes of Feature Types and Feature Instances that are defined in the framework’s Meta Layer.

Because Feature models, especially when using these instance level relationships, allow very flexible structures of Feature relationships to develop during a design process, it is necessary that decompositions, associations, and specifications can be recognised by the modelling system. This is true in particular if the system is to use this interpretation of the relationships in order to find certain semantics in the model. Without having to attempt to interpret the semantics given by a designer to these relationships by means of labels, role names, the system can already distinguish, for instance, which relationships are merely associations, and which are specifications with more detail about the object being modelled.

For a deeper understanding of the various relationships in a Feature model it is necessary to have knowledge about the exact meaning of the individual relationships.
between Features, which can only be accessed by looking at the role names that are given to these relationships by the designer. Automated interpretation, for instance for the purpose of propagating operations on one Feature to other Features that are related with a specific role to that Feature, requires that the role names of relationships are also known in advance to the design system. This implies that classification and standardisation of role names is necessary, an issue that is elaborated in section 8.2.

7.6 Twofold instantiation

The three-layered information infrastructure (see figure 7-4) introduced a double instantiation relationship. Therefore the usage of a consistent terminology is required, to distinguish between the different levels of information definition and the two sorts of instantiation. The first sort of instantiation relationship, following the object oriented paradigm, is found in the model between the Feature Types and the Feature Instances. This is an instantiation relationship seen from the point of view of a designer working with the types and instances. When modelling a design, a designer selects a Feature Type and creates an instance based on the formal definition included in the type. In this perspective, the Feature Instance is an instance of the Feature Type.

However, because Feature Types can be defined 'on the fly' by designers, the structure of the Feature Instances cannot be known in advance. Yet in order to allow a computer-system to work with the new types and instances, the structure of both types and instances must follow certain rules. This is where the second sort of instantiation appears, because the type of content and the type of structure of the Feature Types and Feature Instances must be defined formally. Following the object oriented paradigm again, this is done in the Meta Layer by means of two groups of classes: classes of Feature Types and classes of Feature Instances. As a result, from the viewpoint of the computer-system, the Feature Types are instances of the classes of Feature Types and the Feature Instances are instances of the classes of Feature Instances. Hence, the relationship between the Feature Types and the Feature Instances is no longer an instantiation relationship. Yet, the designer wants to retain this point of view on the instantiation relationship. Therefore the system should maintain the content and structure of the Feature Instances in such a way that they can be presented to the designer as instances of the Feature Types. Internal to the system, however, both Feature Instances and Feature Types are instances of the classes defined in the Meta Layer.

To avoid confusion, the instances from the point of view of the system will be called 'objects', while the instances from the point of view of the designer will retain their name of 'Feature Instances'. However, for both points of view the term instantiation will remain to be used.
A Framework for Modelling Architectural Features

Following the schema in figure 7-5 and the terminology of objects and instances as mentioned, it can be concluded that a Feature Instance is an instance of a Feature Type, but at the same time it is an object of the class of Feature Instances. A Feature Type is an object of the class of Feature Types. Instantiation, thus, takes place in two occasions in the system's point of view: instantiation of a Feature Type into a Feature Instance.

7.7 Classes of Feature Types

The sections 7.7 to 7.9 discuss the classes of Feature Types and classes of Feature Instances that are defined in the Meta Layer of the framework. These classes are presented, using the EXPRESS-G notation technique, in diagrams of four schemata: a schema FeatureTypes, a schema FeatureLibraries, a schema FeatureInstances, and a schema FeatureModels.

Feature Types are the formal representations of domain knowledge. They contain the definition of information concerning a particular concept, a typological aspect of architectural design. Chapter 6 concludes with a proposition for a categorisation of the domain of architectural design on the basis of which Feature Types can be defined. This categorisation concerns the semantic content of the Feature Types from the architectural point of view; it does not concern the content of Feature Types from an information technology point of view. The latter point of view requires different considerations, although the end-result must correspond with the former point of view.

From the information technological point of view, a Feature Type defines what type of content a Feature may have and in what structure it is organised. Also the

---

Figure 7-5 Two kinds of instantiation: from the system's point of view and from the designer's point of view.

Feature Types into an object of this class; and instantiation of a class of Feature Instances into an object of this class. The latter kind of instantiation is the same event of instantiation that takes place in the designer's point of view: instantiation of a Feature Type into a Feature Instance.

---

3 EXPRESS-G is the graphical counterpart of EXPRESS, which is the data-definition language defined in ISO-10303 [ISO TC184 1994c], better known as STEP. The diagrams in this thesis use EXPRESS-G with some minor additions, for instance for the representation of the elements in an 'enumeration' entity. EXPRESS-G and the additions to it are explained in appendix A.
relationships with other Feature Types and therefore the types of relationships that can be found between Features are defined in the Feature Type. From the designer’s point of view, a Feature Type is said to be instantiated into a Feature Instance. Therefore, the classes of Feature Types and the classes of Feature Instances work tightly together.

Six pairs of classes of Feature Types and Instances are defined in the framework. These six classes are designed to cover the information modelling needs that are raised by the categorisation of Feature types in chapter 6 (see figure 6-6 on page 114). The classes consecutively allow the modelling of:

- simple data types
  for modelling simple values in terms of, e.g., string and numbers;
- enumerated data types
  for modelling enumerated symbolic values, e.g. transparent—translucent—opaque;
- complex, or aggregated, types
  for modelling concepts that have a more complex structure, these consist of relationships to other concepts;
- geometric data types
  for modelling concepts that represent a geometric shape;
- constraints
  for modelling constraints on concepts, e.g., the structural dependency between a column and a beam;
- procedural knowledge
  for modelling concepts that represent behaviour; this uses event handling as a basis.

Each of the above kinds of information is represented by a pair of classes for the Feature Type and the corresponding Feature Instance. These are formally defined and can be graphically and textually represented.

**Graphical and textual notation of the classes**

Working with a design support system that is based on the Feature-based framework, it will be the designer who is defining Feature Types. The design support system will then have to deal with the newly defined types of information. For the communication of the Feature Type definitions between designer and system, it is necessary to use representations of them that can be understood by both. As is usual in the area of Information and Communication Technology, a graphical notation technique as well as a

---

4 In section 7.9 it is argued that additional relationships can be defined at the level of Feature Instances.
textual notation technique\(^5\) is used in this research project to represent both Feature Types and Feature Instances. Existing notation techniques have been found inapt for the representation of the specific characteristics of these entities in the Feature-based framework. Along with the definitions of the classes in this and the next section, the graphical and textual notation technique that has been defined for these classes are presented and examples are given.

The textual notation reminds of the declaration of classes in OO languages such as C++. Its syntax is defined using the Wirth Syntax Notation (WSN, see appendix B). Some choices in the syntax definition of the textual notation have been made in an arbitrary, though well considered, manner, such as the choice of valid characters for identifiers in the framework and the representation of dates. This kind of decision is not discussed here at length, since the eventual outcome is not relevant at this stage of the framework development.

The graphical notation is in many ways similar to other graphical notation-techniques, but aims to be sufficiently different in order to avoid confusion with other techniques like the EXPRESS-G notation that is used for the classes in the Meta Layer. The main entities in the graphical notation are rounded boxes representing Features and Feature Types, containing their identifier.

### Basic elements of the notation

The following basic elements in the notation are not further expanded in the syntax specifications: string, number, integer, real, boolean, date, and identifier. In the syntax specifications, they are printed in italics. Their meaning is briefly described here. A string is a character-string containing any sequence of characters, enclosed in double quotes ("'). Double quotes can be included in a string by preceding them with a backward slash (\). A number is any numeric value, including integers and reals. Boolean values are indicated by the keyword boolean. They can be either of the literals true and false. A date is a numeric representation of a date value, in this format: yyyymmdd.

An identifier is a character-string starting with an alphabetic character and containing any alpha-numeric characters and/or any of the following characters: \(\sim\#\%\_\$\).

\(^5\) Examples of textual representations are programming languages such as Pascal, C, Lisp, Prolog, et cetera, and data definition languages such as the EXPRESS language developed in ISO 10303 or STEP. These textual representations are formal and explicit enough to be unambiguously interpreted by computer-systems, and still readable enough for humans. They form the most basic medium of communication between human and computer.

Examples of graphical representations are the notation techniques developed for data modelling methods, such as IDEF, NIAM, EXPRESS-G, and the OO approaches developed by, e.g., Booch, Meyer, Rumbaugh, et cetera. Originally, the purpose of these notation techniques was to support software developers in the early stages of software design. However, they are now evolving to become the main medium in advanced interfaces for software development.
It may contain spaces and line-feeds in the graphical notation, but in the textual notation these are to be omitted or replaced with the underscore character (_).

7.7.1 The base class: FeatureType

The classes of Feature Types are based on an abstract\(^6\) class called FeatureType. This base class defines characteristics applicable to all Feature Types, including their identification, the name of their author, their date of definition, and a description. The base class is graphically presented in figure 7-6.

![Diagram of FeatureType](image)

*Figure 7-6 Diagram 1 of Schema FeatureTypes: Definition of the abstract base class FeatureType. (This is a preliminary diagram, for the eventual diagram, see figure 7-19 on page 154.)*

Feature Types are identified by their typeID, which is an instance of the class TypeID, see figure 7-7. A TypeID is built up using a rather flat structure, which is related to the way Feature Types are to be classified in Feature Type Libraries. The TypeID has a typeName and a sectionID, which in turn has a sectionName and a libraryName. This structure conforms to the categorisation of Feature Types in sections within Feature Type Libraries (see also figure 7-22 on page 158).

TypeIDs of Feature Types are used for reference within the definition of other Feature Types and within the context of Feature Instances in Feature models. Within its scope the TypeID should be unique. This is likely to be a precarious matter, since designers are free to define their own libraries of Feature Types and, at the same time, want to use standardised or commercial Feature Type Libraries. Having computers generate unique IDs is in itself not a complicated issue, yet generating meaningful unique IDs obviously is.

---

\(^6\) An abstract class, in object oriented contexts, is a class that can not be instantiated, i.e. no objects of such a class can be created. The purpose of this kind of class is merely to serve as a super-class for sub-classes that are derived from the super-class, allowing these sub-classes to share common characteristics. A base class is a super-class that itself has no super-classes.
Notation of the abstract base class FeatureType

Since the base class FeatureType is an abstract class, objects of this class cannot be created. Therefore this class does not need a graphical notation. However, the notation of the attributes of the FeatureType class is important, since they form the basis for the notation of all other types. Referring to figure 7-6, the attributes of the FeatureType class are: typeID, typeCreated, typeAuthor, and typeDescr. The typeID is built up from a libraryName, a sectionName, and a typeName.

### Syntax of the attribute typeID of the class FeatureType

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>typeID</td>
<td>sectionID '.' typeName</td>
</tr>
<tr>
<td>sectionID</td>
<td>libraryName '.' sectionName</td>
</tr>
<tr>
<td>libraryName</td>
<td>identifier</td>
</tr>
<tr>
<td>sectionName</td>
<td>identifier</td>
</tr>
<tr>
<td>typeName</td>
<td>identifier</td>
</tr>
</tbody>
</table>

The libraryName and sectionName are included in the typeID to allow a Feature Type to be uniquely identified. In the Graphical notation of the Feature Types, the typeName is sufficient to identify a type, given that all other types within the context of a diagram are defined in the same section and library and the section and library are named in the caption of the diagram. If more than one section is represented in the diagram, the sectionNames need to be included in the typeIDs of those types that are defined in a section different from the section mentioned in the caption of the diagram. The same is true for the level of libraries: types from libraries other than the one mentioned in the diagram's caption need to include the libraryName in their typeID.
In examples, the `sectionName` and `libraryName` are not required in neither graphical nor textual notation, as is demonstrated in most of the examples throughout this chapter. The other attributes of `FeatureType` do not have a graphical notation, but are textually represented as shown below.

**Syntax for the attributes `typeCreated`, `typeAuthor`, and `typeDescr` of the class `FeatureType`**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>typeCreated</code></td>
<td>'TypeCreated' '{' date '}'</td>
</tr>
<tr>
<td><code>typeAuthor</code></td>
<td>'TypeAuthor' '{' string '}'</td>
</tr>
<tr>
<td><code>typeDescr</code></td>
<td>'TypeDescr' '{' string '}'</td>
</tr>
</tbody>
</table>

The general structure for the syntax of a Feature type definition is given below. The attribute `typeBehaviour` is included here in the textual notation, but defined only after the introduction of the subclass `Handler Feature Type` in section 7.7.2 on page 155 (see also figure 7-19 on page 154).

**General syntax for the definition of Feature Types**

```
<type-def> = <simple-type-def> | <enum-type-def> |
   | <complex-type-def> | <geometric-type-def> |
   | <constraint-type-def> | <handler-type-def> |
<standard-body> = <typeCreated> <typeAuthor> <typeDescr>
   | <typeBehaviour> |
```

### 7.7.2 Subclasses of the class `FeatureType`

The diagram in figure 7-8 shows the set of Feature Type classes that can be instantiated. All of these subclasses inherit the attributes of the base class `FeatureType`. There are six subclasses of Feature Types. Simple Feature Types can be used for the definition of Feature Types that represent simple data types. Enumeration Feature Types are used for the definition of a data type that is defined by a list of names, enumerating the possible values that instances of that data type may assume. The class of Complex Feature Types allows the definition of structures of Feature Types, using the various kinds of relationships as discussed in section 7.5. Geometric Feature Types are defined for the specific representation of geometric data. Constraint Feature Types provide the ability to define constraints on the values of Feature Instances and on the relationship between Feature Instances. Handler Feature Types, finally, are used for the definition of behaviour, they will be used to attach behaviour to the level of Feature Types and the level of Feature Instances as well.
In the remainder of this section, these various subclasses of FeatureType are discussed in more detail and their notation and examples are presented.

**The class SimpleFeatureType**

This class facilitates the definition of Feature Types that represent simple data such as integers, reals, character strings, and boolean data. The type of data that the Feature Type defines is specified in the value of the baseType enumerated attribute. Every Simple Feature Type must specify its base type. A unit for the data may be specified if relevant, e.g. \( m^2 \) or \( W/m^2 \), and a default value for the simple data can be included. Simple
Feature Types may also specify a domain for the values that instances may assume. Domains and defaults are discussed in detail in section 7.7.3.

Figure 7-9 Excerpt from Diagram 3 of Schema FeatureTypes (see figure 7-8 on page 136):
Definition of the class SimpleFeatureType.

**Notation of the class SimpleFeatureType**

The four different simple data types that can be defined each have, for reasons of legibility, a distinct notation, both graphical and textual. The general syntax for a Simple Feature Type definition is as follows:

<table>
<thead>
<tr>
<th>General syntax for the definition of Simple Feature Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>simple-type-def = string-type-def</td>
</tr>
<tr>
<td>real-type-def</td>
</tr>
</tbody>
</table>

The graphical notation for Simple Feature Types with base-type string consists of a rounded box with the typeId of the Feature Type and a letter S in a square on the left.

<table>
<thead>
<tr>
<th>Syntax for the definition of string-based Simple Feature Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>string-type-def = 'string' typeId '{' string-body '}' .</td>
</tr>
<tr>
<td>string-body = standard-body</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>string-domain-decl = 'TypeDomain' '{' string-domain '}' .</td>
</tr>
<tr>
<td>string-default-decl = 'TypeDefualt' '{' string '}' .</td>
</tr>
</tbody>
</table>

Domain and default for Simple Feature Types are not graphically represented. The syntax for domains can be found in section 7.7.3.
Example of a string-based Simple Feature Type

```plaintext
string Briefing.SpatialPlan.SpatialFunction {
    TypeCreated (19980704)
    TypeAuthor ("Jos van Leeuwen")
    TypeDescr ("Function of a space")
    TypeDefault ("Living")
}
```

For reasons of brevity and clarity, in the remaining examples the standard-body part of the definition, including the date, author, and description of a type, is left out. Also, the libraryName and sectionName, in the above example Briefing and SpatialPlan respectively, are omitted. For the other Simple Feature Types, the syntax is defined similarly, as is shown in the following.

```plaintext
Example of a integer-based Simple Feature Type

```plaintext
integer NumberOfHinges {
    TypeDomain (2,3,..5)
    TypeDefault (3)
}
```

Example of a real-based Simple Feature Type

```plaintext
real Area {
    TypeDomain ([0,->>)
    TypeDefault (25.9)
    TypeUnit ("m2")
}
```
Classes of Feature Types

### Syntax for the definition of boolean-based Simple Feature Types

<table>
<thead>
<tr>
<th>boolean-type-def</th>
<th>boolean-body</th>
<th>boolean-domain-decl</th>
<th>boolean-default-decl</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>boolean</code> typeID <code>'( boolean-body ')'</code></td>
<td><code>standard-body [ boolean-domain-decl ]</code></td>
<td><code>TypeDomain '(' boolean-domain ')'</code></td>
<td><code>TypeDefault '(' boolean ')'</code></td>
</tr>
</tbody>
</table>

### Example of a boolean-based Simple Feature Type

```java
boolean IsExterior {
    TypeDefault {true}
}
```

---

**The class EnumerationFeatureType**

A simple data type that cannot be defined using the class SimpleFeatureType is the enumeration. Enumerations are data types that serve to identify a single selection from an enumerated list of names. The Enumerated Feature Type includes an ordered set of character strings denoting the identifiers that instances of the enumeration may assume as their value. A default value for the enumeration type can be included, as well as a domain for the value of instances. A domain for this Feature Type, that already specifies a range of possible values, seems redundant or even irrelevant, but is included for completeness. It may bear relevance in cases where a (temporary) limitation on the enumeration is required.

![Diagram](image.png)

**Figure 7-10** Excerpt from Diagram 3 of Schema FeatureTypes (see figure 7-8 on page 136): Definition of the class EnumerationFeatureType.

---

**Notation of the class EnumerationFeatureType**

The class of Enumeration Feature Types is graphically noted by a letter E in the square of the symbol. Its textual notation is similar to those of the Simple Feature Types, but includes the ordered set of enumerated identifiers preceded by the label TypeItems. In the graphical notation, the enumerated identifiers are not necessarily made visible, but can be indicated by listing all identifiers and connecting this list with the enumeration symbol using an arrow line. The syntax for domains can be found in section 7.7.3.


### Syntax for the definition of Enumeration Feature Types

```plaintext
enum-type-def = 'enum' typeID '{' enum-body '}' .
enum-body = standard-body enum-decl [ enum-domain-decl ]
enum-decl = 'TypeItems' '{' enum-items '}' .
enum-items = identifier (',', identifier) .
enum-domain-decl = 'TypeDomain' '{' enum-domain '}' .
enum-default-decl = 'TypeDefault' '{' identifier '}' .
```

### Example of an Enumeration Feature Type

```
enum TypeOfWall {
  TypeItems {Single, Cavity, Compound}
  TypeDefault {Compound}
}
```

---

**The class ComplexFeatureType**

Information that is in any sense structured in a more complex manner needs to be defined using the class ComplexFeatureType. This class defines a simple composition structure using a set of Components. A Component is a reference to another Feature Type; Complex Feature Types do not contain other Feature Types (compare object attributes in OO approaches), but merely refer to other Feature Types. This restriction is very valuable for the resulting flexibility of information models, mainly because it allows Feature Types to share components. However, this approach also introduces some potential complications, in particular for the integrity management of the Feature Types and Instances, for example when Feature Types are modified. This issue is further discussed in section 8.4. A similar structure of references is found at the instance level of Complex Features, which allows Feature Instances to share components. An example of such a structure is given in figure 7-31 on page 167.

The usage of ComplexFeatureType can lead to both tree-like (hierarchical) and lattice-like information structures. The contents and structure of Components are discussed in detail in the next subsection and presented in figure 7-12. A default for the Complex Feature Type can be specified, which refers to an instance within the same library-section; the existence of instances in Feature Type Libraries is addressed in a separate discussion in section 7.7.4. A Complex Feature Type also has an optional domain for the instances of the type, referring to a set of instances within the same library-section (see also the discussion and diagrams for defaults and domains in section 7.7.3).

A ComplexFeatureType optionally relates to a SuperType, which refers to the TypeID of a Feature Type forming the ComplexFeatureType's supertype. This establishes single inheritance of Complex Feature Types. One restriction is in effect here: only ComplexFeatureTypes can be the supertype of a ComplexFeatureType.
This restriction follows the fact that Complex Feature Types can themselves not contain data in the way for instance a SimpleFeatureType contains data. ComplexFeatureTypes only contain references and can therefore not inherit from, e.g., SimpleFeatureTypes. Other Feature Types than ComplexFeatureTypes are included in the 'inheritance-tree' by reference through the components. A Complex Feature Type that has a supertype inherits all the components of its supertype and those of its supertype's supertypes.

The support-class Component

Complex Feature Types (objects of the class ComplexFeatureType) are decomposed into objects of the supporting class Component. Each component of a Complex Feature Type has a role indicating the kind of relationship the component has with the Complex Feature Type. The role is specified by the roleType (an enumerated value indicating a decomposition, association, or specification type of role; note that specialisation is not a valid role type, this relationship is modelled using the superType attribute of a ComplexFeatureType) and identified by its rolename, which is a string of characters.

Components are either TypeComponents or InstanceComponents. TypeComponents are components that are declared at the type level and given values at instantiation time. The actual data that define these components are specified during instantiation and stored in a Feature model, using the Component class of the featureInstances schema (see figure 7-29). At the type-level, the TypeComponents merely declare the structure of the Complex Feature Type. An example of the usage of a TypeComponent is the declaration of a ComplexFeatureType called material which has a TypeComponent called colour. This declaration specifies that all materials have a colour, without providing any value for the colour.

In a Complex Feature Type, TypeComponents can have multiple occurrences, which, at the type level, may be limited in number by means of the cardinality. The cardinality of a component specifies a minimum and maximum number of occurrences that a component should have. How the modelling system should deal with these
minimum and maximum numbers during a design session is discussed further in section 8.3 on instantiation issues. If the cardinality of a component is not specified, this means that the cardinality is \([0..1]\), which is to say that the component is a single, but optional, component. Components that are not supposed to be optional must specify the cardinality; for single components this means a cardinality of \([1..1]\).

The values that a component can assume are specified by its domain, and it can also be given a default value. Note that both domain and default override, in the context of the Complex Feature Type, the domain and default defined for the Feature Type referred to by the component. However, the contents of both domain and default must, of course, be in accordance with the Feature Type the component refers to (the different kinds of domains and defaults are presented in section 7.7.3).

An example to illustrate the function of the cardinality in a component is the relationship of a wall with several elements in the wall. The actual elements in a specific wall are instances of a component in the Complex Feature Type defining the wall. This component refers to the TypeID of the Feature Type defining the elements; it has a role with, for instance, the rolertype Decomposition and a rolename 'element'; and it has a cardinality that specifies a minimum of 0 elements with no upper limit: \([0..\infty]\).

![Diagram](image_url)
This example is worked out in more detail on page 145 where the graphical and textual notation of the definitions of Complex Feature Types is presented.

InstanceComponents, as opposed to TypeComponents, do not merely declare the structure of a Complex Feature Type, they also define the values that are contained in the components of this structure. This means that the value of such a component is defined for all instances that are to be created from the particular Complex Feature Type. For example, a Complex Feature Type called wooden beam would have a component of a type called material, but this component should always represent the characteristics for the material wood. Therefore this component does not refer to the TypeID of a Feature Type called material, but rather to the FeatureID of the Feature Instance called wood of the Feature Type material. This instance must be stored in a Feature Type Library, since it is referred to directly in the definition of Feature Types. Thus, Feature Type Libraries do not only contain definitions of Feature Types, but also definitions of Feature Instances that have significance at the type-level\(^7\). These Feature Instances can also be referred to at the instance level. More on this aspect is found in section 7.7.4. InstanceComponents, like TypeComponents, can refer to multiple Feature Instances.

Because this part of the definition of the Complex Feature Type represents an invariant information structure, the cardinality for this relationship need not be defined. Although the type of the Feature Instances that participate in an InstanceComponent is not specified for the component, they should all be of the same type.

Components in a Complex Feature Type can refer to any other class of Feature Type, including the Geometry, Constraint, and Handler Feature Types. These three types, which are defined further on in this section, may accept parameters for their functionality and behaviour. It is by means of these parameters how information is communicated from the Complex Feature Type and its properties to, for instance, a geometry component that is to represent that Feature Type.

For example: a Complex Feature Type called Door may be defined with a specification by two components h and w of the types Height and Width respectively, and an association to a type DoorGeometry that represents the door geometrically. The DoorGeometry must somehow be related to the door's height and width, which is done by passing the components h and w as parameters to the DoorGeometry component. This example is elaborated after the definition of the Geometric Feature Type on page 149.

Parameters can be passed at the instance level or at the type level. Passing parameters at the type level means that the role names of components of a Complex Feature Type are passed as parameters and that all instances of that Complex Feature

\(^7\) This aspect of the framework is to some extent comparable to the usage of, for instance, static data in class declarations in C++.
Type use this way of passing parameters. The above example shows this approach.
Another form of passing parameters at the type level is to pass identifiers of Feature
Instances created at the type level as parameters. This form is necessary, for instance,
when constant values must be passed at the type level; Feature Instances are required to
provide these constant values. The issue of Feature Instances at the type level is further
discussed in section 7.7.4.

Passing parameters at the instance level is discussed in section 7.8.2 on page 168.
Parameters passed at the instance level override parameters that are passed at the type
level. The attribute parameter is defined on page 169.

Notation of the class ComplexFeatureType

<table>
<thead>
<tr>
<th>Syntax for the definition of Complex Feature Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>complex-type-def = 'complex' typeID ['(' super-typeID ')']</td>
</tr>
<tr>
<td>super-typeID = typeID</td>
</tr>
<tr>
<td>complex-body = standard-body [ complex-domain-decl ]</td>
</tr>
<tr>
<td>complex-domain-decl = 'TypeDef' ['{' complex-domain '}']</td>
</tr>
<tr>
<td>complex-default-decl = 'TypeDefault' ['{' featureID '}']</td>
</tr>
<tr>
<td>decomp-decl = 'Has' component-decl</td>
</tr>
<tr>
<td>assoc-decl = 'Assoc' component-decl</td>
</tr>
<tr>
<td>spec-decl = 'Spec' component-decl</td>
</tr>
<tr>
<td>component-decl = ( type-component-decl</td>
</tr>
<tr>
<td>type-component-decl = typeID roleName [ cardinality ]</td>
</tr>
<tr>
<td>inst-component-decl = roleName ['{' integer '}'] '=' featureID</td>
</tr>
<tr>
<td>domain = string-domain</td>
</tr>
<tr>
<td>default = string</td>
</tr>
<tr>
<td>,...</td>
</tr>
</tbody>
</table>

The syntax for the class ComplexFeatureType provides the possibility to include, in
parentheses after the identifier of the type, the typeID of the super-type for the defined
Feature Type, denoting a Specialisation relationship. The relationships to the
components of the Complex Feature Type are textually denoted in the body of the
notation by three keywords; Has, Assoc, and Spec, for respectively Decomposition,
Association, and Specification relationships. These relationships are further declared by
specifying the typeID of the component, its role name, and optionally its cardinality,
domain, and default.

Modelling Architectural Design Information by Features
The term featureID, used for the defaults and instance components in a Complex Feature Type, is defined in section 7.8.1 on page 162. The term param_list is defined in section 7.8.2 on page 169.

Before an example of a Complex Feature Type is presented, the graphical notation of the different relationships need to be defined. Relationships are shown by a line connecting the two related Feature Types. The relationships are to be interpreted bidirectional, yet one direction is emphasised and given a roleName. The emphasised direction of a relationship is graphically marked by a symbol at the end of the line. This symbol also indicates the type of the relationship as is shown in figure 7-13.

Specialisation relationships can be abbreviated by joining the two symbols of supertype and subtype, as shown in figure 7-14. The Feature Type at the beginning of the line is also the type that, in the textual notation, contains the relationship as a component. Yet, in the case of a specialisation, it is the subtype at the end of the line that declares the relationship in the textual notation. Note that specialisation relationships can only be modelled through the supertype-subtype construction of Complex Feature Types.

### Example of a Complex Feature Type

```plaintext
real Area {
    TypeDefault (0.0)
}

complex Space {
    Spec Area area {{12,->}} = 27.5;
}

complex Room(Space) {
    Assoc Wall enclosedBy[0..?];
}

complex Wall {
    Has WallElement element[0..?];
}

complex WallElement {
}
```

In the above example, the Complex Feature Type Space has a specification relationship to the type Area with the role named area. The domain for this
specification is such that the minimum area for the space is 12, and its default is 27.5. Both domain and default are given for the component area to override any domain and default defined in the type Area.

Note that the Feature Type WallElement is a Complex Feature Type that has no components. This is to demonstrate and remind that the definition of Feature Types is not a determined process, but is an ongoing activity during the design process. The details regarding the composition of the wall elements will be specified in later stages during design. Perhaps the type WallElement will be replaced by another, readily detailed Feature Type. This dynamic manipulation of the information structure in a conceptual model of Feature Types is further discussed in section 8.4.

The support class DeclaredParameterList

The three classes of Feature Types that remain to be defined, Geometric, Constraint, and Handler Feature Types, all use parameters to provide either external or internal processes with access to the information in a Feature model. In the case of Geometric Feature Types, they provide an external, parametric, geometric modeller with the parameters required to generate the desired geometries. Handler Feature Types contain procedures that require access to the Features available in a model. This communication of Feature data to those processes is done using lists of parameters. The support class DeclaredParameterList is defined to allow those lists of parameters to be declared; it is used to specify the type of parameters that will be used by the various processes.

A DeclaredParameterList is an ordered list of parameter declarations. Ordered, because the order of the parameters in this list is important for passing them to the accessing process. Parameter declarations consist of a name for the parameter and the identifier of the Feature Type that the parameter must be an instance of. If this type is not specified, then any type of Feature Instance may be passed for this parameter; the parameter is then untyped. Figure 7-15 shows the definition of the support class DeclaredParameterList. The list of parameters should at least contain one parameter, but may be of variable length, meaning that the parameter of the last declared type may be passed repeatedly. If this is the case, the flag variableList is set to True.

![Diagram 7-15](image_url)
Classes of Feature Types 7.7

Notation of declared parameter lists

Declared parameter lists are used by Geometric Feature Types, Constraint Feature Types, and Handler Feature Types. The textual notation of a declared parameter list is in parentheses after the typeID of the Feature Type it belongs to. Each parameter declaration is shown by the Feature Type of the parameter and the name of the parameter. If the type is not significant, it is replaced by the keyword ANY. A parameter list can be of variable length, which is indicated by an ellipsis after the last parameter (at least one parameter must be declared in a parameter list). This is not graphically represented.

In the graphical notation, parameters are shown as relationships to the Feature Types of the parameters, with an association symbol. The name of the parameter is written next to the relationship line. Untyped parameters are indicated using the symbol shown on the right.

<table>
<thead>
<tr>
<th>Syntax for the declaration of parameter lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>decl-param-list = param-decl (',', param-decl)</td>
</tr>
<tr>
<td>param-decl = ( 'typeID'</td>
</tr>
<tr>
<td>param-name = identifier .</td>
</tr>
</tbody>
</table>

The class GeometricFeatureType

The class for Geometric Feature Types is included here but not yet detailed. A relationship with some form of parametric geometry is predictably necessary. However, how this parametric geometry should be defined has not been part of this research project; it is merely represented in the framework by a string indicating the type of geometry, e.g. sphere, box. At this point in the project, it is simply assumed that a parametric geometry modeller module will become a part of the design support system, and that this module can access the Geometric Features and their parameters in the model. The basis for such a module could be found in commercially available geometric modelling engines such as Parasolid8 and ACIS9.

Figure 7-16 Excerpt from Diagram 3 of Schema FeatureTypes (see figure 7-8 on page 136): Definition of the class GeometricFeatureType.

8 Parasolid is a trademark of Parametric Technology Corporation (www.ptc.com).
9 ACIS is a trademark of Spatial Technology (www.spatial.com).
The types of the parameters that the parametric geometry requires, are defined by the attribute parameterList of the Geometric Feature Type, which is a list of parameter declarations. The actual passing of parameters is done either at the instance level by means of a Geometric Feature Instance, or at the type level where a Geometric Feature Type is referenced as a component in a Complex Feature Type, by means of the inclusion of a parameter list for that component (see also page 143).

**Notation of the class GeometricFeatureType**

The graphical notation of Geometric Feature Types includes a letter G in the square box on the left of the symbol. In the textual notation, the type of geometry is indicated in the body of the definition, using the keyword TypeGeometry. The notation of the declared parameter list is already discussed separately.

```
G Geometric Feature Type
```

**Syntax for the definition of Geometric Feature Types**

```
geometric-type-def = 'geometry' typeId '(' [ decl-param-list ] ')' '
{ ' geometric-body ' } '.
geometric-body = standard-body geometry-type .
geometry-type = 'TypeGeometry' '{ ' identifier ' } '.
```

**Example of a Geometric Feature Type**

```
G IPEgeometry
```

The above example shows the notation of a Geometric Feature Type called IPEgeometry. It represents a parametric geometry that is indicated by the name IPE and declares two parameters, one of the Feature Type Height, with the name profileheight, another of the Feature Type Length, with the name length. This assumes that there exists a parametric geometry with the name IPE that takes two parameters, specifying the height and length of the IPE profile, and that has embedded knowledge about the other dimensions of the profile, based on the given height parameter.
A second example shows how Geometric Feature Types can be related to from within a Complex Feature Type.

![Diagram of Example](image)

This example demonstrates how the components beamheight and beamlength of the Complex Feature Type IPEBeam serve as parameters to the component beamGeom. Graphically this relation cannot be expressed, the diagram only shows the declaration of the parameters of IPEgeometry and its role as a component in the IPEBeam type.

The definition and syntax of passing parameters is described in section 7.8.2 on page 169.

**The class ConstraintFeatureType**

Constraints form the basis for the relationship of information models with constraint solving and constraint checking modules of design support systems. Constraints have been applied in design to model and maintain restrictions on the geometry of designed objects.

One approach to applying constraints in geometric modelling situations, is to use temporal interval algebra. Allen [1983] defines five relationships between a point and an interval: ahead, front-touch, in, back-touch, and behind. This approach has been used in [Kelleners et al. 1997; Kelleners 1999] to develop an object-oriented constraint solver that handles interval-interval constraints that can be applied on box geometries. The solver recognises the following five types of constraints: connection constraints (e.g. touch, align sides), distance constraints, a contains constraint, a non-intersection constraint, and unary constraints (e.g. fixed position, fixed side, fixed dimension, fixed orientation, minimum dimension, maximum dimension). The latter are not interval constraints but necessary for the solving process.

Dohmen [1998] describes a system that implements constraint-based Feature validation. The system handles constraint types such as attach constraints, which allow the attachment of geometric Features to one another, and semantic constraints for topologic properties, comparable to the connection and non-intersection constraints by Kelleners. Furthermore, Dohmen deals with geometric constraints, specifying e.g. the position, or distance, of Features relative to each other. Algebraic constraints relate more
general parameters of Features, for instance specifying dimensional proportions. Dimension constraints restrict the domain of Feature parameters.

![Diagram of Constraint Feature Type](image)

**Figure 7-17** Excerpt from Diagram 3 of Schema Feature Types (see figure 7-8 on page 136): Definition of the class `ConstraintFeatureType`.

Although the technology of constraint solving as such is not a part of this research project, their integration in design support systems is anticipated by the definition of the Constraint Feature Types. In this project, a constraint is identified in a Constraint Feature Type by the attribute `constraint` which is represented as a string. The parameters used by the constraint are declared by the list of parameter declarations that is represented by the attribute `parameterList`. In principle, relating the Feature model to constraints in this manner is not restricted to constraints on geometric data only, but is open to specifying constraints on any other type of data as well. It allows constraints to be applied on any other design aspect, such as structural design or construction planning. The sole restriction is that the constraint can be reduced to some form of mathematical equation or set of equations for which a solver is available. Two such applications of Constraint Feature Types are clear:

- Implementation of 'derived attributes' of a Feature, e.g. the calculated outcome of an algebraic equation with reference to various locations in the Feature model. An alternative approach for implementing a 'derived attribute' is by determining its value using an procedure in a Handler Feature Type (see below).
- Automated management of integrity and consistency of the information in a Feature model. Rules for integrity management that can be expressed using algebraic equations can be modelled through Constraint Feature Types. More complicated procedures require the Handler Feature Types.

**Notation of the class ConstraintFeatureType**

The graphical notation of Constraint Feature Types includes the letters Ct in the square box on the left of the symbol. The string representing the constraint is shown in the textual notation using the keyword `TypeConstraint`. The declared parameter list is noted textually and graphically as has been discussed before.
Classes of Feature Types

Syntax for the definition of Constraint Feature Types

<table>
<thead>
<tr>
<th>Constraint Feature Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraint-type-def = 'constraint' typeID '(' [ decl-param-list ] ')'</td>
</tr>
<tr>
<td>constraint-body = standard-body constraint-type</td>
</tr>
<tr>
<td>constraint-type = 'TypeConstraint' '(' identifier ')'</td>
</tr>
</tbody>
</table>

Example of a Constraint Feature Type

```
constraint ConnectSpaces (Space space1, Space space2) {
    TypeConstraint (adjacentTo)
}
```

The class HandlerFeatureType

Event handling will be an important mechanism in design support systems. Its purpose is to incorporate procedural knowledge in both the conceptual model (the Feature Types) and the actual models (the Feature Instances). Using event handlers, procedural knowledge can be captured in models concerning, e.g., instantiation of types, user-interaction, validation, problem-solving, and model-evolution.

```
Figure 7-18 Excerpt from Diagram 3 of Schema FeatureTypes (see figure 7-8 on page 136): Definition of the class HandlerFeatureType.
```

Handler Feature Types, as presented in figure 7-18, include a procedure that is executed in response to the occurrence of an event. It is beyond the scope of this thesis to specify in which form the procedures are implemented and how they are executed by design support systems. However, some of the required functional capabilities of these procedures can be deduced from the following issues. Information must be passed to the procedure concerning the context in which it is called. For this purpose, a list of parameters is declared for the Handler Feature Type (see the attribute parameterList in figure 7-18 and its definition in figure 7-15). Besides the information passed through this list to the procedure of an event handler, event handlers need to have ‘knowledge’ about their direct environment, such as the Feature Instance it is attached to (this is the Feature Instance that notifies the event handler that the event
has occurred). This also includes the relationships of this Feature Instance, both typological and instance level relationships (see section 7.9). To have knowledge about its environment, in this case, means that from within the evaluated procedure, data can be accessed that is located in the notifying Feature Instance, and data can be retrieved from the relationships of this notifying Feature Instance. The procedure should have access also to data that is not directly related to the notifying Feature Instance, but that is located elsewhere in the Feature model. This capability provides a way for event handlers to manifest themselves as 'active' relationships between Feature Instances in a model. It allows, for instance, modifications to a Feature Instance to have an effect on other parts of the Feature model.

Most likely, also knowledge external to the Feature model is relevant for the evaluation of the procedure of an event handler. The integration of specific applications with a Feature modelling system, for instance a cost calculation module, would require access to external data, such as catalogues, price-lists, and other cost related information.

Although its importance is well acknowledged, the exploration of the possibilities of event handling mechanisms in a design support system is not elaborated within the scope of this research project. Some of the envisioned applications of event handling include the following.

- Modelling dependencies between parts of a Feature model by means of event handlers. An example is the instantiation, modification, or deletion of Features as a result of the occurrence of a particular event. Section 8.3 discusses handled instantiation in more detail.
- Implementation of 'derived attributes' of a Feature, e.g. the calculated outcome of an algebraic function with input from various locations in the Feature model. An alternative approach for implementing a 'derived attribute' is by determining its value using an equation in a Constraint Feature Type. Handler Feature Types are expected to introduce additional functionality for those cases where a procedural approach for the determination of the derived attribute's value is required.
- Automated management of integrity and consistency of the information in a Feature model. Here, a similar consideration can be made: rules for integrity management that can be expressed using algebraic equations can be modelled through Constraint Feature Types. More complicated procedures require the Handler Feature Types.
- Integration of external information sources in a Feature-based design support system, leading to enhancement of the issues mentioned above. For this application of Handler Feature Types, their procedure must be allowed to access the external sources.

**Notation of the class HandlerFeatureType**

The graphical notation of Handler Feature Types includes a letter H in the square box on the left of the symbol.
Classes of Feature Types

7.7

Syntax for the definition of Handler Feature Types

<table>
<thead>
<tr>
<th>handler-type-def</th>
<th>handler-body</th>
<th>handler-procedure</th>
</tr>
</thead>
</table>
| 'handler' typeID (' [ decl-param-list ] ')
	(' handler-body ')' )
| 'standard-body handler-procedure' |
| Procedure ' (' code ')' ) |

The declared parameter list of the Handler Feature Type is optional and included in parentheses after the typeID attribute. The term code is not further specified, because the syntax and semantics of the language of procedures have not been defined in this project. In the example below, the code is replaced by comments, explaining how the procedure will perform.

Example of a Handler Feature Type

```
handler ModifyAssoc (ANY associates) {
    TypeAuthor "Jos van Leeuwen"
    Procedure {
        // modify all instances that are somehow
        // associated to the notifying instance
        // and that are specified by the
        // 'associates' parameter.
    }
}
```

Scenarios for using Handler Feature Types

Three different scenarios are foreseen in the framework, in which event handlers are applied in a model. Two types of events are distinguished in these scenarios. In the first two scenarios, the event is triggered by a Feature Instance, signifying for instance a modification of the instance that should be reacted upon by the event handler. This kind of event is called a Feature-event. The third scenario deals with events that are triggered by the modelling system itself, for instance events signifying that an evaluation task has been started. These events are called System-event. Events are enumerated identifiers that are not further specified in the scope of this project.

Scenarios 1 and 2 for modelling event handlers: handling Feature-events

In scenarios 1 and 2, an event handler is attached to a Feature Type, which is called the notifying Feature Type. This can be any Feature Type, therefore the definition of the abstract base class FeatureType requires redefining. In the new definition of the class FeatureType, which is shown in figure 7-19, a Feature Type can have a set of EventHandlers. An EventHandler identifies the event it handles and is identified in the context of the notifying Feature Type by the attribute handledName. The difference between the scenarios 1 and 2 lies in the way parameters are passed to the event handler.
In scenario 1, the parameters are defined at the level of the notifying Feature Type, meaning that all instances use the same parameter definitions. These parameter definitions are provided by means of a single Handler Feature Instance that is attached to the notifying Feature Type. In this scenario, the handler attribute of the EventHandler entity in figure 7-19 contains the FeatureID of a Handler Feature Instance. This scenario is useful only when all handling of the event for this Feature Type should, at any time, be passed the same parameters. However, it should be noted that a parameter may refer to the role of a component of a complex Feature Instance, which provides certain flexibility. In case the parameters differ per instance of the notifying Feature Type, scenario 2 must be followed.

In scenario 2, each instance of the notifying Feature Type has its own definition of parameters and therefore its own Handler Feature Instance. At the type level, the TypeID of the Handler Feature Type is specified by the handler attribute in the definition of the notifying Feature Type (see figure 7-19). At the instance level, each notifying Feature Instance has a relationship to a Handler Feature Instance that specifies the parameters. This relationship is an attribute of the class FeatureInstance which is defined in section 7.8.1.

**Scenario 3 for modelling event handlers: handling System-events**

The third scenario deals with System-events that are triggered by the modelling system rather than by Features. In this scenario, event handlers are not attached to Feature Instances, but are modelled by themselves: the Handler Feature Instances exist independently in the Feature model. The parameters that are specified in such a Handler Feature Instance do not depend on a notifying Feature. Examples for the three scenarios are given in section 7.8.2 on page 173.

![Diagram 1 of Schema FeatureTypes: Definition of the abstract base class FeatureType.](figure.png)

*Modelling Architectural Design Information by Features*
Notation of the attribute \texttt{typeBehaviour} of the class FeatureType

The relationship of an event handler with a notifying Feature Type is defined using the attribute \texttt{typeBehaviour}, as shown in figure 7-19. This attribute includes either the \texttt{typeID} of the Handler Feature Type, or the \texttt{featureID} of the Handler Feature Instance, depending on how the parameters are to be passed. Furthermore, it includes the role name for the event handler in the context of the notifying Feature Type and the identifier for the event that is handled. The inclusion of the \texttt{typeBehaviour} attribute in the definition of Feature Types is already anticipated in the textual notation for the attributes of the class FeatureType in section 7.7.1. Its syntax is further defined as follows.

\begin{verbatim}
Syntax for the attribute \texttt{typeBehaviour} of the class FeatureType

\begin{verbatim}
typeBehaviour = 'Behaviour' '(' ( event-decl ) ')' .
  event-decl = ( typeID | featureID ) roleName
    ' [' eventID ']' ; ;
  eventID = identifier .
\end{verbatim}
\end{verbatim}

\end{verbatim}

Examples of how to model the relationships between notifying Feature Types and Instances and Handler Feature Types and Instances are given in section 7.8.2 on page 173, after the definition of the class HandlerFeature.

7.7.3 Domains and defaults

Feature Types, as has been described in the previous sections, may specify \texttt{domains} and \texttt{defaults} for their instantiation. The definitions of the various kinds of domains are presented in figure 7-20 and the different kinds of defaults are found in figure 7-21.

For Simple Feature Types and Enumeration Feature Types the domain specifies what values are valid for the instantiated Features. The default specifies the initial value a Feature will get during instantiation. The contents of the SimpleDomain and SimpleDefault for Simple Feature Types depend on the baseType (and possibly the unit) of the Feature Type. For example, a Feature Type Area may specify a minimum area of 12.5 square meters, with a default value of 17. A SimpleDomain is either a set of character strings, a selection of boolean values, or a numeric domain, which is a set of numeric ranges. Some possibilities of numeric ranges are included in the diagram: a single numeric value; a discrete range, i.e. a succession of numeric values; and a continuous range with an upper and lower bound. Note that in the EXPRESS language, of which the graphical counterpart EXPRESS-G is used in the diagrams, a Number data type includes both the Integer and Real data types. This means that occurrences of the Number data type can be either integer or real values.
Figure 7-20 Diagram 5 of Schema Feature Types: Definition of Domains.

Enumeration Feature Types have an EnumDomain and EnumDefault that select from the enumerated identifiers that are defined within the Enumeration Feature Type.

For Complex Feature Types, the issue of domains and defaults is rather more complicated. A ComplexDefault for the Complex Feature Type as a whole specifies an instantiation of the Feature Type complete with instantiated components. This Complex Feature Instance must be included in the same Feature Type Library section as the Complex Feature Type. When, during a modelling session, such a Complex Feature Type is instantiated, the default action will be to create a reference to the Feature Instance included in the Feature Type Library. Similarly, a ComplexDomain specifies a set of Feature Instances that are included within the same library-section. Instantiating a Complex Feature Type with such a domain is restricted to selection of a reference from this set of Feature Instances in the Feature Type Library.
Classes of Feature Types

Figure 7-21 Diagram 6 of Schema Feature Types: Definition of Defaults.

Notation of Domains

The syntax for domains varies per class of Feature Type. Five types of domains are defined: string-domains for Simple Feature Types with base type string, numeric-domains for Simple Feature Types with base type integer or real, a boolean-domain for base type boolean, enum-domains for Enumeration Feature Types, and complex-domains for Complex Feature Types.

### Syntax for domains

<table>
<thead>
<tr>
<th>Domain</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>string-domain</td>
<td><code>string [ ';' string ]</code></td>
</tr>
<tr>
<td>numeric-domain-term</td>
<td><code>numeric-domain-term [ ';' numeric-domain-term ]</code></td>
</tr>
<tr>
<td>numeric-domain</td>
<td>`discrete-domain</td>
</tr>
<tr>
<td>discrete-domain</td>
<td><code>number </code></td>
</tr>
<tr>
<td>continuous-domain</td>
<td>`('&lt;.'</td>
</tr>
<tr>
<td>boolean-domain</td>
<td><code>boolean [ ';' boolean ]</code></td>
</tr>
<tr>
<td>enum-domain</td>
<td><code>identifier [ ';' identifier ]</code></td>
</tr>
<tr>
<td>complex-domain</td>
<td><code>featureID [ ';' featureID ]</code></td>
</tr>
</tbody>
</table>

The term `featureID` is defined in section 7.8. Some examples of these domains are given hereafter to clarify their meaning further.
Examples of the notation of Domains

<table>
<thead>
<tr>
<th>Examples</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>string: &quot;living&quot;; &quot;cooking&quot;</td>
<td>The string value may only be picked from the ones listed.</td>
</tr>
<tr>
<td>numeric: 2, 3, 5, 10,..100</td>
<td>Values may be 2 or 3 or in the range 5 to 100 in steps of 5, including 5 and 100.</td>
</tr>
<tr>
<td>&lt;4,0]; [2, &gt;&gt;</td>
<td>Values may be any real number between -4 and zero, but not including -4, or greater than or equal to 2.</td>
</tr>
<tr>
<td>boolean: true</td>
<td>The value will always be true.</td>
</tr>
<tr>
<td>enum: single; double</td>
<td>The enumerated value may be one of these identifiers.</td>
</tr>
<tr>
<td>complex: hto3; hto5; dstA3</td>
<td>Listed are the IDs of the Features that may be selected in the context of this domain.</td>
</tr>
</tbody>
</table>

7.7.4 Feature Type Libraries

The structure of Feature Type Libraries, as has been briefly indicated already in the beginning of this section, is a rather flat, hierarchical structure. Feature Types are categorised into sections in Feature Type Libraries. The assumption that such a flat hierarchy should suffice is based on the observation that most classification systems in use at present do not provide more levels. However, using this means of categorisation in practice will need to confirm this. If necessary, the structure of Feature Type Libraries must be amended to allow for nested sections of Feature Types. The initial structure of Feature Type Libraries is presented in figure 7-22, showing a decomposition of a Feature Type Library into sections which in turn contain the Feature Types.

![Diagram of Feature Type Library](image)

Figure 7-22 Diagram 1 of Schema FeatureLibraries: Definition of FeatureTypeLibrary. (This is a preliminary diagram, for the eventual diagram, see figure 7-23.)

At various occasions in the above sections, it has become clear that the inclusion of Feature Instances in Feature Type Libraries is a necessity. The reasons for this are summarised below.
• **defaults and domains for Complex Feature Types**
The defaults and domains for Complex Feature Types are instances that need to be accessible from within the Feature Type Library and therefore are included in the library;

• **defaults and domains for components of Complex Feature Types**
The same is true for the defaults and domains of the type-components of Complex Feature Types. If type-components refer to Complex Feature Types, these specify the initial and valid instances of those Complex Feature Types within the context of the Complex Feature Type that contains the references. Again the instances need to be accessible from within the Feature Type Library;

• **instance-components of Complex Feature Types**
Besides type-components, Complex Feature Types can also contain instance-components, which are not references to Feature Types, but references to Feature Instances. These instances must also be available in the Feature Type Libraries.

• **Handler Feature Instances specifying the parameters for event handlers attached to any Feature Type**
As discussed during the definition of Handler Feature Types, and referring to scenario 1 of how to use these types, a Handler Feature Instance can be used in the definition of any Feature Type to specify the parameters for handling events that occur in relation to the Feature Type. The Handler Feature Instance, for this purpose, must be available with the definition of the Feature Type, i.e. in the Feature Type Library.

• **Feature Instances passed as parameters at the type level**
For the passing of constant values to components of Complex Feature Types at the type level, it is necessary that Feature Instances are available at the type level to provide these values.

---

**Figure 7.23 Diagram 1 of Schema FeatureLibraries: Definition of FeatureTypeLibrary.**
(This diagram supersedes the diagram in figure 7.22.)
This leads to a revised structure of Feature Type Libraries as displayed in figure 7-23. Besides Feature Types, a section within a Feature Type Library now also contains Feature Instances. The class Feature is defined in section 7.8.1.

7.8 Classes of Feature Instances

While Feature Types are the formal representation of domain knowledge, defining typological aspects of architectural design, Feature Instances are the formal representation of information concerning a particular design. What kind of information a Feature Instance contains and in what structure, is defined by a Feature Type of which the instance is instantiated. Analogous to the classes of Feature Types presented in section 7.7, this section presents the classes of Feature Instances that define the format for the Feature Instances that can be created within the framework.

![Diagram 1 of Schema Feature Instances: Definition of the abstract base class Feature. (This is a preliminary diagram, for the eventual diagram, see figure 7-37 on page 177.)](image)

7.8.1 The base class: Feature

The classes of Feature Instances are based on the abstract class Feature. The class defines that all Feature Instances contain the date and author of their instantiation, as well as an optional description. A graphical representation of the definition of the base class Feature is found in figure 7-24. As the name of this class indicates, the term Feature Instance is abbreviated to Feature, both terms are used interchangeably in the rest of this chapter.
The Feature Type that a Feature is instantiated from is referred to in the Feature by its typeID. In the schema FeatureInstances this is a reference to the schema FeatureTypes. Features are identified by their featureID, represented in figure 7-25. FeatureIDs are to be unique within their context, which is either a particular Feature model or a section in a Feature Type Library. A Feature Instance can be included in a Feature Type Library, as discussed in section 7.7.4, for reference in the definition of Feature Types or for reference from within Feature models. The FeatureID of a Feature Instance that is included in a Feature Type Library consists, besides a string, of the sectionID indicating the section in the library that contains it.

Any Feature Instance may define certain behaviour, which is modelled by means of the behaviour attribute (see figure 7-24). This attribute refers to a list of event handlers, which either have been defined at the type level of the Feature, or are added for a particular instance only. The event handler specifies the identifier of the Handler Feature that is to handle the event. In the case of an event handler defined at the type level, the event to be handled is specified at the type level. For handlers that are added at the instance level, the event must be indicated by the event attribute of the handler. Finally, each handler has a role name in the context of the Feature Instance.

Notation of the abstract base class Feature

The abstract base class Feature does not need a graphical or textual notation, because objects of this class cannot be created. Its attributes, however, are inherited by its subclasses and therefore need a textual notation. These attributes are, as shown in figure 7-24, featureID, typeID, created, author, descr, and behaviour. The attribute typeID specifies the type of which this Feature is an instance. This attribute is defined earlier on page 134.
A Framework for Modelling Architectural Features

Syntax for the attributes featureID, created, author, descr, and behaviour of the class Feature

| featureID = [ sectionID '::*' ] instanceName |
| instanceName = identifier |
| created = 'Date' '({' date ')}' |
| author = 'Author' '({' string ')}' |
| descr = 'Descr' '({' string ')}' |
| inst-behaviour = 'Behaviour' '({' (event-handler ) ')}' |
| event-handler = featureID roleName '([' eventID ')]' |

The attribute featureID consists of an identifier that is to be unique within its context, normally a Feature model. However, Feature Instances may also be included in Feature Type Libraries, in which case the ID also includes the sectionID (i.e. library name and section name) to indicate where they are located. The sectionID is followed by two colons and the name of the Feature Instance.

General syntax for the notation of Feature Instances

| feature-inst = simple-inst | enum-inst | complex-inst |
| geometric-inst | constraint-inst | handler-inst |
| standard-inst-body = created author descr |
| inst-behaviour inst-relations |

The attribute behaviour refers to the set of event handlers that are associated with the particular Feature Instance. An event handler is specified by the event identifier and by the identifier of the Handler Feature Instance that is associated with the event. See also the description of Handler Feature Instances in section 7.8.2 on page 171.

The inst-relations element of the standard body of an instance notation refers to an attribute of the class Feature that is not yet presented. It is used to describe relationships that are not defined at the type level, but that are added only for the particular Feature Instance. This aspect of instantiation is further discussed in section 7.9 on instance level relationships.

7.8.2 Subclasses of the class Feature

The classes of Feature Instances that can be instantiated are all subclasses of the abstract class Feature, inheriting all its characteristics. They are represented in the diagram in figure 7-26 and discussed separately in the remainder of this subsection.
The class **SimpleFeature**

The class **SimpleFeature** is the class of Feature Instances that are instantiated from Feature Types that have been defined using the class **SimpleFeatureType**. The instances of this class contain a single, simple value, such as integers, reals, and strings. What actual type of data it is that a particular Simple Feature Instance contains, is defined by the Simple Feature Type, which is referenced in the `typeID` property defined in the super class Feature from which this class inherits (see figure 7-24). Also the unit of the data, e.g. m² or W/m², is defined in the corresponding Simple Feature Type, as are the domain for the data and the default value (see the definition of the class **SimpleFeatureType** in figure 7-8 on page 136).

---

10 The fact that for Simple Feature Instances the `typeID` attribute should refer to the `typeID` of a Simple Feature Type, and not of any other Feature Type, is implicitly assumed here. Such restrictions would clearly have to be built in to an implementation of the framework, but including them in the schemata presented here would not particularly contribute to the clarity of the work. Similar assumptions are made throughout the rest of the discussion of the framework with respect to the kind of identifiers that are relevant and valid in particular contexts.
Figure 7-27 Excerpt from Diagram 3 of Schema Feature Instances (see figure 7-26 on page 163): Definition of the class SimpleFeature.

Notation of the class SimpleFeature

The graphical notation of Feature Instances is similar to the notation of Feature Types. Feature Instances are represented by rounded, grey boxes with the name of the Feature Type, followed by a colon and the name of the Feature Instance; sectionIDs are not graphically represented. For Simple Feature Instances, alternatively, the name of the Simple Feature Instance can be replaced by the value of the Simple Feature Instance. In the example below, the name of the instance room23function is replaced by its value "Pantry". The graphical notation of Simple Feature Instances shows a letter S, I, R, or B in the square on the left of the rounded, grey box to indicate the base type of the instance.

Syntax for the representation of Simple Feature Instances

```plaintext
simple-inst = typeID featureID '=' '{' simple-inst-body '}'
simple-inst-body = standard-inst-body 'Value' '{' value '}'
value = string | number | boolean
```

Example of a Simple Feature Instance

```plaintext
Briefing_Plan.SpatialFunction room23function = {
    Date (19980704)
    Author ("Jos van Leeuwen")
    Descr ("Function of room 23")
    Value ("Pantry")
}
```

What data type constitutes the value of a Simple Feature Instance depends on the baseType attribute of the defining Simple Feature Type. This cannot be read from the notation of the Simple Feature Instance.

For reasons of brevity and clarity, the standard-inst-body part of Feature Instances is omitted in the remaining examples in this and the following chapters. Also
the sectionID (library name and section name) of typeIDs is omitted for the same reason.

**The class EnumerationFeature**

The class EnumerationFeature defines the objects that are instances of Enumeration Feature Types. The value of an Enumeration Feature is an identifier, a string, selected from the range defined by the enumeration of the particular Enumeration Feature Type (see figure 7-8 on page 136). Again, the domain and default for this value is defined in the corresponding Feature Type.

![Diagram of Enumeration Feature](image)

**Figure 7-28 Excerpt from Diagram 3 of Schema Feature Instances (see figure 7-26 on page 163): Definition of the class EnumerationFeature.**

**Notation of the class EnumerationFeature**

The graphical notation of Enumeration Feature Instances shows a letter E in the square on the left of the rounded, grey box.

![Notation of the class EnumerationFeature](image)

**Syntax for the representation of Enumeration Feature Instances**

```
enum-inst = typeID featureID "='"( enum-inst-body )'
enum-inst-body = standard-inst-body
  'EnumValue' '{' identifier '}'
```

**Example of an Enumeration Feature Instance**

```
WallType WallA3type = {
  EnumValue [Single]
}
```

In the textual notation, the value of the Enumeration Feature Instance is noted by the identifier selected from the enumeration defined by the Enumeration Feature Type. As an alternative to showing the name of the enumeration instance, the identifier selected for the value of the instance can be shown in the graphical notation, yet enclosed in single quotes as shown in the example.

**The class ComplexFeature**

Objects of the class ComplexFeature are instances of Complex Feature Types. They form structures of components that refer to other Feature Instances. The components
have a role within the context of the Complex Feature. A component may refer to more than one Feature Instance, in which case those instances share the same role in their relationship to the Complex Feature Instance. The relationship of a component to a Complex Feature Instance is defined at the conceptual level, i.e. in the definition of the corresponding Complex Feature Type. Therefore, the role name of the component corresponds to the role name that was specified in the Complex Feature Type and the number of instances that can share the same role is specified by the cardinality of the component in the type's definition (see figure 7-12 on page 142).

The component structures of Complex Feature Types are used to model three kinds of relationships: decompositions, associations, and specifications. Inheritance relationships, or specialisations, are modelled using the super-subtype structure of Complex Feature Types. The supertype of any Complex Feature Type is always again a Complex Feature Type. As a result, a Complex Feature Instance inherits the relationships defined by its type's supertype. Therefore, the component structure of a Complex Feature Instance also includes the components defined in the Complex Feature Types that are found above its type in the hierarchy of specialised Complex Feature Types, see figure 7-30.

Notation of the class ComplexFeature

The graphical notation of Complex Feature Instances shows a rounded, grey box without additional symbols, containing the name of the Feature Type and the name of the instance. The relationships to its components, i.e. to other Feature Instances, are shown in both the textual and graphical notation using the role name and, if the cardinality is other than 1, the index of the related Feature Instance within the particular
component. Components that are not instantiated are not shown in either notation. This aspect of incompletely instantiated Features is discussed in section 8.3.

### Syntax for the representation of Complex Feature Instances

- **complex-inst**: `typeID featureID = { 'complex-inst-body' }`
- **complex-inst-body**: `standard-inst-body { component-item }
- **component-item**: `roleName [ '{ integer' } ] = 'featureID'`

### Example of a Complex Feature Instance

**Area Area1**

```plaintext
Area Area1 = {
    Value (29.7)
}

Room Office1 = {
    enclosedBy[1] = WallB;
    enclosedBy[2] = WallC;
    usableFloorArea = Area1;
}

Wall WallB = {
    element[1] = ElementB1;
}
```

**Room Office1**

```plaintext
Room Office1 = {
    enclosedBy[1] = WallB;
    enclosedBy[2] = WallC;
    usableFloorArea = Area1;
}
```

**Wall WallB**

```plaintext
Wall WallB = {
    element[1] = ElementB1;
}
```

**NB**. The textual notation of the above example is not shown completely.

The graphical representation of another example is shown in figure 7-31. This example demonstrates the effect of defining components of Features as references. Components of a Complex Feature Instance are not contained in the instance, but are relations, references, to other Feature Instances. This allows Feature Instances to share components. In the example, the 'function' component of both rooms 21 and 22 refer to the same Feature of the type Function that has the value 'Office'.

**Figure 7-31** Shared Features made possible through the reference structure of the components of Complex Features.
The support class Parameter

The last three subclasses of Feature Instances use parameters to pass information from the Feature model to either parametric geometry, constraints, or event handlers. These subclasses are the classes of Geometric, Constraint, and Handler Feature Instances. They are defined next in this section. Parameters can be passed in four forms:

1. identifiers of Feature Instances;
2. components of Complex Feature Instances;
3. instance level related Feature Instances;
4. references to Feature Instances.

These four forms determine the definition of the support class Parameter which is shown in figure 7-32. Because parameters are passed in ordered lists, their position in the list must correspond to that of their declaration in the declared parameter list of the Feature Type (see figure 7-15 on page 146).

In the first form, where identifiers of Feature Instances are passed as parameters, the attribute featureID of a parameter simply contains the FeatureID of that Feature Instance. This provides access to the information available in the Feature Instance, such as the value of a Simple Feature Instance.

If the Feature Instance is a Complex Feature Instance, the component attribute of the parameter may refer to one of the components of that Complex Feature Instance by specifying the role name of the component; this is the second form of parameter passing. If the component refers to multiple Feature Instances, which depends on the cardinality of the component as defined in the Complex Feature Type, an index is required to indicate the exact Feature Instance that is to be passed as parameter. If that index is not specified, this means that, by way of the role name, all Feature Instances referred by the component are passed as the parameter.

Because relationships between Features can also be defined at the instance level, as discussed in section 7.9, in the third form of parameter passing, the role name refers to such an instance level relationship. Again this is done by specifying the role name of the relationship. Any Feature Instance, not just Complex Feature Instances, can have such instance level relationships.

Figure 7-32 Diagram 4 of Schema FeatureInstances: Definition of the support class Parameter.
The last form of parameter passing involves references to Feature Instances in the context of the parameters. For all three kinds of Feature Instances that use parameters, a parameter may refer to that Feature Instance itself by specifying the value SELF for the attribute reference. This attribute replaces the featureID attribute; they are mutually exclusive. For Handler Features, a parameter may refer to the notifier of the event handler by using the value NOTIFIER for the attribute reference.

The usage of the reference attribute is demonstrated after the definition of Handler Features. In the schema in figure 7-32, the attributes featureID and reference are shown as optional attributes, while in fact they are mutually exclusive.

**Notation of parameter lists**

Lists of parameters are textually noted as comma-delimited lists. Each parameter consists of the featureID of the Feature Instance that is to be passed as the parameter, or one of the keywords SELF and NOTIFIER as appropriate.

In case a parameter should refer to a component of a Complex Feature Instance or to an instance level relationship, the role name, preceded by a period, and the optional index in square brackets follow the identifier of the Feature Instance. This is done in both the textual and graphical notation.

<table>
<thead>
<tr>
<th>Syntax for the notation of parameter lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>param-list = parameter { ',' parameter }</td>
</tr>
<tr>
<td>parameter = ( featureID</td>
</tr>
<tr>
<td>component-param = { '.' roleName [ '{ integer' ] }</td>
</tr>
</tbody>
</table>

In graphical notations, a parameter is shown as an association relationship from the Feature Instance that uses it to the Feature Instance that is passed as parameter. The name of the parameter is shown next to the relationship line. If the parameter list is declared as a variable length list, then the parameters that have the same name are shown using an index in square brackets behind the parameter name, for example oneOrMoreDoors[3] indicates the third parameter to be passed with that name.

The references SELF and NOTIFIER are shown using Complex Feature symbols, regardless of the actual type of Feature. They are not resolved, i.e. not replaced by the referred Feature Instances. The symbol of the Feature Instance passed as parameter, if this is a Simple Feature Instance, may show the value of the simple data rather than its identifier.

Examples are given after the definition of the classes of Feature Instances that use parameters, next in this section.

**The class GeometricFeature**

Instances of a Geometric Feature Type merely consist of relationships to the Feature Instances that act as parameters for the parametric geometry defined in the Geometric
Feature Type. The parameters form an ordered list that corresponds to the ordered list of parameter declarations in the Geometric Feature Type. In the framework, it is assumed that any explicit representation of the geometry is generated from the data available in the Feature model and has a repository that is, both transient and persistent if desired, external to the Feature model. The relationships between the external geometric model and the Geometric Feature Instances in the Feature model are to be maintained by the geometric model.

Figure 7.33 Excerpt from Diagram 3 of Schema FeatureInstances (see figure 7.26 on page 163): Definition of the class GeometricFeature.

Notation of the class GeometricFeature

In the graphical notation, the relationships of the Geometric Feature Instance with its parameters are shown using the association relationship symbol accompanied by the name of the parameter. In the textual notation, the parameter list is enclosed in parentheses and follows the identifier of the Geometric Feature Instance. The keyword SELF may be used to make a parameter refer to the Geometric Feature itself.

Syntax for the representation of Geometric Feature Instances

| geometric-inst = | typeID featureID ' (' [ param-list ] ') ' '='  |
| geometric-inst-body = | standard-inst-body |

Example of a Geometric Feature Instance

used type definition:

```text
TypeGeometry (IPE)
geometry IPEgeometry (Height profileheight, Length length) {
  TypeGeometry (IPE)
}
IPEgeometry IPE240
(Height240, BeamLengthBC) = {
}
```

The class ConstraintFeature

While Constraint Feature Types define constraints and their required types of parameters, Constraint Feature Instances only need to specify which Feature Instances actually provide the values for these parameters. The identifiers of these Feature
Instances are inserted in the ordered list of parameters, which corresponds to the ordered list of parameter declarations defined by the Constraint Feature Type. As with the parameters of Geometric Feature Instances, the parameters may actually be referring to components of Complex Feature Instances, in which case the role name and possibly an index are required.

![Diagram](image)

**Figure 7-34 Excerpt from Diagram 3 of Schema Feature Instances (see figure 7-26 on page 163): Definition of the class ConstraintFeature.**

### Notation of the class ConstraintFeature

Similar to the Geometric Feature Instances, the relationships to the parameters of Constraint Feature Instances are shown with association relationship symbols in the graphical notation. The textual notation shows the parameter list in parentheses after the featureID of the Constraint Feature Instance. The keyword SELF may be used to make a parameter refer to the Constraint Feature itself.

```plaintext
constraint - inst = typeID featureID "(' [ param-list ] ')" ;
constraint-inst-body = standard-inst-body .
```

### Example of a Constraint Feature Instance

```plaintext
used type definition:
constraint ConnectSpaces {Space space1, Space space2} {
    TypeConstraint {adjacentTo}
}
ConnectSpaces KitchenToDining
(Kitchen, Diningroom) = {
}
```

### The class HandlerFeature

A Handler Feature Instance provides the parameters that are to be passed to the procedure defined by the Handler Feature Type that it is an instance of. The parameters are provided in a list, similar to those of Constraint and Geometric Feature Instances.

Handler Features can be attached to a Feature Type or a Feature Instance, to handle so-called Feature-events. When such a Feature-event is triggered, the system is notified that the procedure defined by the Handler Feature Type should be called.
passing it the parameters specified in the Handler Feature Instance. Handler Features can also be modelled to handle so-called System-events, in which case they are not attached to a notifying Feature Type or Feature Instance. The identifier of the System-event must be specified for this kind of Handler Feature Instance (see the definition in figure 7-19 on page 154).

![Figure 7-35 Excerpt from Diagram 3 of Schema FeatureInstances (see figure 7-26 on page 163): Definition of the class HandlerFeature.]

**Notation of the class HandlerFeature**

In order to define the notation of Handler Feature Instances, the three scenarios described after the definition of the class HandlerFeatureType are followed (see page 153).

<table>
<thead>
<tr>
<th>Syntax for the representation of Handler Feature Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>handler-inst = typeID featureID [' [ 'eventID ' ] ' ]</td>
</tr>
<tr>
<td>' [ ' [ param-list ] ' ] ' ' = '</td>
</tr>
<tr>
<td>'{ ' handler-inst-body ' } '</td>
</tr>
<tr>
<td>handler-inst-body = standard-inst-body .</td>
</tr>
</tbody>
</table>

In the first scenario, a Handler Feature is attached to a Feature Type, specifying the parameters for all event handling that should occur for the instances of that Feature Type. In this scenario, the handler should have access to the notifying Feature Instance and to its directly related Feature Instances. Because the Handler Feature is attached at the type level, the identifier of the notifying Feature is not yet known. Therefore the keyword NOTIFIER is introduced in the definition of parameter lists for handlers, which may substitute the featureID in both the graphical and textual notation of parameters. Note that the keyword SELF may be used to refer to the Handler Feature itself in order to access its instance level relationships. The eventID in this scenario is not relevant, because the event to be handled is specified by the notifying Feature Type.
Classes of Feature Instances

Example of a Handler Feature Instance, attached to a Feature Type (scenario 1)

```plaintext
ModifyAssoc doorModifier
(NOTIFIER.door) {
}

complex Wall {
    Behaviour {
        modifyDoors [onModified];
    }
    Assoc Door door[0..?];
}
```

In the second scenario, a Handler Feature is attached to a Feature Instance, specifying the parameters for the event handling of that instance only. Although the featureID of the notifying Feature is now known, the same keyword NOTIFIER may be used to refer to that Feature.

Example of a Handler Feature Instance, attached to a Feature Instance (scenario 2)

```plaintext
Wall wallB {
    door[1] = doorB1;
    door[2] = doorB2;
    Behaviour {
        modifyDoors [onModified];
    }
}
```

The third scenario describes the handling of so-called System-events. This is done by Handler Features that are not attached to a Feature Instance or Feature Type.

Example of a Handler Feature Instance for handling of a System-event (scenario 3)

```plaintext
used type definition:
handler CostEvaluator
(BuildingBlock blockToEvaluate, ...) {
    Procedure {
        // evaluate the costs related to the buildingblock passed as the 'blockToEvaluate' parameter(s).
    }
}

CostEvaluator evaluateCosts_1
(onStartEvaluate) (blockC, blockD) = {
}
```

In the above example, the parameter blockToEvaluate of the handler is specified by directly assigning the identifiers of the blockC and blockD instances. An alternative approach, which effectively does the same thing, is to use instance level
relationships to relate both BuildingBlock instances to the Handler Feature. For the parameter, the role name of this relationship (block) can now be used, together with the keyword SELF to refer to the Handler Feature itself. This example illustrates the usage of instance level relationships, which are discussed in the section 7.9.

Example of a Handler Feature Instance for handling of a System-event (scenario 3)

<table>
<thead>
<tr>
<th>blockToEvaluate</th>
<th>SELF::block</th>
</tr>
</thead>
<tbody>
<tr>
<td>CostEvaluator:</td>
<td>evaluateCosts 2</td>
</tr>
<tr>
<td>block[1]</td>
<td></td>
</tr>
<tr>
<td>block[2]</td>
<td></td>
</tr>
<tr>
<td>BuildingBlock:</td>
<td>blockC</td>
</tr>
<tr>
<td>blockD</td>
<td></td>
</tr>
</tbody>
</table>

7.8.3 Feature Models

Features are organised into Feature models. The organisation of a Feature model, as shown in figure 7-36, is defined as a simple collection of Features. Feature models are identified by the ModelName attribute. A consequence of this simple organisation of Features is that any subdivision of the model into, for instance, hierarchical levels of info, must be devised in another manner. The advantage of this approach, which is very much related to the objectives of Feature modelling that are being aimed at in this project, is of course that any subdivision can be made accordingly to the designer's intent.

![Figure 7-36 Diagram 1 of Schema FeatureModels: Definition of FeatureModel](image)

For such a subdivision, special Feature Types must be instantiated to model the backbone of the structure to which all Features in the model are to be attached. For example, if a subdivision of the model is requested into the levels of 'building', 'storeys', 'spaces', 'building elements', 'components', etc., Feature Instances with interrelationships will be modelled to represent these levels. Subsequently, all Features describing information at the level of storeys will be modelled with a relationship to the 'storeys' Feature. These relationships are preferably modelled at the instance level only in order to
Instance level relationships

If a relationship is desired between Feature instances that is not defined at the typological level, a problem arises because the relationship cannot be instantiated. The following example will be used to discuss the options for solving this problem: A designer wants to model the relationship between doors and an escape route through the building. This implies that certain doors require adequate fire resistance. In the conceptual model used by this particular designer, it happens that fire resistance is not defined as part of the Feature Type Door.

Three different actions that can be taken to solve this problem are discussed below:

1. modifying the type definition;
2. creating a new type or sub-type;
3. modelling the relationship for the particular instance only.

All three of these options need to be provided by design information systems based on the Feature modelling framework.

The first option is to change the definition at the typological level, adding the desired relationship to the Feature Type. This solution is acceptable if all instances of the particular type require this relationship. Making the relationship optional increases the chances that this is a satisfactory situation, however, in many cases the desired relationship bears no relevance to other instances of the same type: for many doors fire resistance may not be significant. Adding to the definition of a Feature Type, its size and complexity continue to increase and eventually it will lose its relevance, which is being the common denominator of its instances.
A second option is to define a new Feature Type, either as an amended copy of the original Feature Type, or as a Feature sub-type that inherits the super-type’s characteristics while adding the desired relationship: a Fire_Door-type is defined. This approach avoids the problem of growing Feature Types. However, the definition of a new Feature Type for each divergent instance may result in an unmanageable number of Feature Types (security doors, automatic doors, etc). Furthermore, an instance can only be derived from one type: these door-types are exclusive so that fire doors cannot be attached to the security system. Apart from this, it may not be the designer’s notion of the situation that ‘adding a relationship to a particular entity’ is regarded as the creation of a new concept.

The actual situation in the third option may well be described as a designer adding a non-typical relationship to a Feature instance (the fire resistance concerns only this door), without regarding this as a change in concept of the original Feature Type. This situation should be modelled exactly in this manner, for the information model to represent most accurately the rationale of this design decision.

What is required thus, is a sort of incidental relationship that can be modelled between Feature instances without the necessity of including the relationship in the definition of the Feature Type. For this purpose we distinguish a type-relationship from an instance-relationship. Type-relationships are defined at typological level and have possible relevance to all instances of a type, whereas instance-relationships are added at instance level without being defined at typological level, thus without relevance to other instances of the same type. In the diagram, the roles of instance-relationships are shown in italics.

Three of the four categories of relationships described in section 7.5, are relevant also for the kind of relationship that can be expected at instance level. Table 7.3 lists the seven types of relationships that are part of the FBM infrastructure. The three types of instance-relationships are semantically equivalent to their counterparts at the typological level.
One of the most important implications of instance-relationships for information modelling systems concerns the way information is searched for and addressed in the information model. Instead of using knowledge from the structure of the conceptual model (the typological level), the system must now search for relationships at instance level as well in order to find the requested information in a model. For instance, information concerning costs may be available in the model by means of relationships defined for the different types of elements in a building, but in addition, certain costs may be added, as a specification, to particular instances of building elements that bring additional costs to the construction process: ‘all steel columns of the type HE230A cost $x.xx per meter, but the one labelled D23 costs an additional $z.zz because it is harder to position.’

![Diagram of Schema FeatureInstances: Definition of the abstract base class Feature.](image)

### 7.9.1 Implications for the framework

The purpose of instance level relationships is to allow unforeseen relationships or non-typical relationships to be added to Feature Instances. These relationships cannot be restricted by the definition of the Feature Type of the instance in question and must be
allowed for all kinds of Feature Instances, not just Complex Feature Instances. This implies that instance level relationships need to be included in the definition of the base class Feature. The revised definition of the class Feature is represented in the diagram in figure 7-37. The revised Feature class defines the possibility of adding relationships between Feature Instances and giving these relationships a role. This structure is similar to the component structure defined in Complex Feature Types. The InstanceRelation entity in the diagram specifies the role of the instance level relationship, and has a list of Feature identifiers that refer to the related Feature Instances. This allows an instance level relationship to exist with multiple Feature Instances, using the same role name, similar to the way this is possible with components in a Complex Feature Type. Although the type of the instance level relationship is not pre-defined, the modelling system should ensure that the Feature Instances that participate in one such relationship should be of the same type.

**Notation of instance level relationships**

In the textual notation, instance level relationships are noted in a similar way as the definition of relationships at the type level. The same three keywords are used for the definition of the three kinds of relationships, Spec, Has, and Assoc. The relationship is further defined by the featureID of the related Feature Instance, the role name of the relationship, and an optional integer in square brackets, which indicates the index of the related Feature for the specified role name. Because there is no definition of this relationship at the conceptual level, there also is no definition of its cardinality, domain, or default. In fact, this means that the cardinality of instance level relationships is always \[0..\]. If an instance level relationship needs to be added using the same role name as another instance level relationship, both will need to be given the additional integer to distinguish between them. In both the textual and graphical notation, instance level relationships are shown with the role name in italics.

<table>
<thead>
<tr>
<th>Syntax for the attribute instRelation of the class Feature</th>
</tr>
</thead>
</table>
| \[
| \text{inst-relations} = ( \text{inst-spec-decl | inst-decomp-decl | inst-assoc-decl} ) .
| \text{inst-spec-decl} = 'Spec' \text{inst-relation-decl} .
| \text{inst-decomp-decl} = 'Has' \text{inst-relation-decl} .
| \text{inst-assoc-decl} = 'Assoc' \text{inst-relation-decl} .
| \text{inst-relation-decl} = \text{roleName} [ ' [' \text{integer} ' ] ' ] '=' \text{featureID} ';' .
| \] |

Examples of defining Feature Types and modelling Feature Instances and instance level relationships are given in section 8.9 on the application of the FBM framework in an architectural design case study (page 213).
Implementation of the FBM Framework in Architectural Design

The final chapter in this thesis discusses a variety of issues related to the implementation, both technical and organisational, of the FBM framework in architectural design. Starting with the issue of how to define a Feature Type and its relationships, it subsequently addresses the processes of instantiating Features and manipulating both Feature Types and Instances, discussing different strategies for these procedures that can be followed by designers. The possible role of the FBM approach in the efforts on standardisation and communication is discussed in comparison with the developments in STEP and the IFCs. In relation with the developments of design systems in the VR-DIS research programme, the aspect of interfacing metaphors for Features and Feature Types is discussed, and the proposed modular architecture of the VR-DIS development, based on the FBM framework, is presented. The last two sections in this chapter discuss the work on prototyping the FBM framework and on a design analysis case study that was performed using one of the resulting prototypes.
8.1 Strategies for Feature Type definition

8.1.1 Identification of a concept

Defining a Feature Type follows the decision to formalise a design concept. Therefore the first problem to address in Feature Type definition is: how to recognise and identify a concept? Two different points of view from which to approach this problem are discussed:

1. design domain knowledge and vocabulary;
2. strategies for Object Oriented classification.

The first point of view discusses how concepts can be acquired from sources of design knowledge, whereas the second point of view presents the most common approaches in OO Analysis to classify a given knowledge domain. Both points of view must be considered in the processes of identifying concepts for the formalisation of design knowledge.

Design domain knowledge and vocabulary

The first point of view in the quest for concepts is taken from the body of knowledge in the domain of design. This knowledge, particularly in the complex discipline of architectural design, is not always readily available or easily accessible. Certain concepts in this body of knowledge are scientifically defined, such as the SI units (meter, second, Kelvin, Volt, etc.). These often are rather elementary concepts, which is to say that, in terms of information structure, they do not bear much complexity. Other, perhaps more complex concepts may be defined in a less exact manner, but still be well conceived, such as industrial products of which all characteristics are known and available from manufacturers. The terminology used for these products and their characteristics forms the basis for defining the Feature Types that are to represent this kind of concept. Perhaps the most important kind of concept in design, especially in early stages, is the kind that forms the core of architectural design theory and methods. These concepts represent elements of design that can be either concrete or abstract, tangible or intangible, exact or indeterminate. For this kind of concept, the vocabulary of the design domain may be a suitable starting point for their formal definition into Feature Types. This vocabulary, in architecture, is not formally defined, but many terms have traditional meanings that are generally accepted.

The first consideration in the process of identifying a concept should be whether a term exists that covers the potential concept. Terms are normally used to indicate the names of, e.g., systems, structures, products, materials, functions, organisational units. An analysis of the way the term is used should be projected onto the concept being identified and reveal if the term actually represents that concept or not. If an existing, accepted term cannot be found, there are four possible consequences:
Strategies for Feature Type definition

- The potential concept needs some adjustment to fit a term that is reasonably close to describing the concept.
- The potential concept covers a combination of multiple terms.
- The potential concept introduces a new term in the design vocabulary.
- Any combination of the three options above.

Whether or not new terminology should be defined involves a trade-off between aspects such as:

- Acceptability of the concept in the design discipline. This may be an important issue when the concept serves, e.g., purposes of standardisation, regulation, or information exchange.
- Desired or allowed level of ambiguity of the defined concept. Because new terminology, as opposed to traditional terminology, is not naturally known, its introduction may result in various interpretations of the term, which have to be verified against the concept's definition. Any ambiguity in the formal definition of the concept will then allow variance in the interpretation of the term.
- Completeness and exactness of the definition. As a result of the previous aspect, the completeness and exactness of the definition of the concept cannot rely on knowledge inherently related to traditionally known terminology.
- Uniqueness of the concept in relation to existing vocabulary. Using new terminology allows a concept to be defined distinctly and independently from implicit meanings related to existing terminology. This can be a prerequisite when the uniqueness of the concept is to be stressed or when distinction from other concepts is necessary.

Closely related to design domain knowledge are the areas of design methodology and design theory. Design methodology, according to Roozenburg and Eekels [1995], is the science that studies the structure, methods, and rules of design. Design methodologies are either developed while focusing on the design process as a whole, e.g. [Hall 1995], [Pahl and Beitz 1988], and [French 1985], or intended for specific domains or phases in design. An example of the latter, given by Roozenburg and Eekels, is the morphological method, which relates characteristics and functions of a design with the variant components for that design in an array containing all conceivable solutions. For the identification of design concepts, it is interesting to look at the subjects used in specific design methods, especially those subjects that form an intrinsic part of the method.

A recently developed methodology for architectural design is presented as Generic Representations by Achten [1997]. This methodology involves an approach to the identification of design content in architectural graphic representations. Its hypothesis is that graphic representations made during the design process imply the design decisions that are made. The research project shows how it is possible to extract...
such design decisions from the graphic representations, by inferring the declarative knowledge embedded in these representations. The methodology proposed by Achten involves using generic representations, and the design knowledge acquired from them, as a model for procedural decision making in design.

Many design methods, like the example given above, develop design aids, such as archetypes, design patterns, proportional or other measuring systems, rules for design schemata for instance for floor plans and elevations, and so on. These tools can be regarded as the design concepts that are applied in the context of the design situation at hand, using established procedures from the design method. The definitions of these concepts are not always clear and explicit but may involve implicit knowledge about the usage and meaning of the concepts themselves and of the procedures for using them in design. The formalisation of this kind of concept into a Feature Type requires that all relevant knowledge be made explicit, which may involve formalisation of other concepts and knowledge about concepts that have not been identified explicitly before. Classification strategies will help to identify these.

**OOA strategies for classification**

The term *classification* in Object Oriented Analysis refers to the task of the software engineer to identify classes of objects in the domain for which software is to be developed. These classes then form the backbone for the design of procedures and data storage of that software. According to Sowa [1984] there have only been three general approaches to classification:

- **Classical categorisation.**
  The criteria for sameness of objects is formed by their properties: objects that have one or more properties in common belong to a category.

- **Conceptual clustering.**
  First, the conceptual descriptions of classes are formulated, then objects are classified according to these classes using a 'best fit' method.

- **Prototype theory, or classification by example.**
  The class is not defined conceptually, but by means of an example: a prototype. Objects are member of the class only if they sufficiently resemble the prototype.

In practice of Object Oriented Analysis, these approaches are combined and/or followed sequentially. Classification forms the main starting-point not only for the identification but also for the design of object classes. It supports the determination of structures of classes and of the structure of data and behaviour of these classes. As such, these approaches to classification are valuable also during the definition of Feature Types.
For the process of Feature Type definition, the identification of the concept in its context is important, not the way how this is achieved. Once the terminology for the concept is determined the definition of the concept itself can start.

8.1.2 Definition of the Feature Type

Once a concept to be formalised has been identified and subjected to the considerations described above, the actual definition of the Feature Type for that concept can commence. Definition of a Feature Type is a procedure that is very similar to the definition of object classes in OO approaches for which many strategies and checklists have been described [Booch 1994; Meyer 1990; Rumbaugh 1991]. Aspects that need to be considered when defining a Feature Type are the following:

- **bottom-up versus top-down**
  In the complex structure of information that embodies a concept, where should the formalisation process be started: at the most abstract notion of this concept, or at its most detailed characteristics? This aspect needs to be considered not only for the definition of a single Feature Type, it should also be part of the overall strategy for defining libraries of Feature Types. A bottom-up approach has the advantage of stimulating the re-use of existing Feature Types, such as those in the Generic Feature Type libraries. A top-down approach, however, allows the designer to think in logical hierarchies that are common in architecture.

- **typical versus non-typical**
  Is the information to be formalised typical for the concept, or does it merely concern a characteristic of a particular instance of that concept? This has to do with the reusability of the concept: when too much information is included in the Feature Type, then the reusability of the concept will be less: perhaps some less common characteristics should not be defined as part the Feature Type, but modelled as instance level relationships for particular Feature Instances only.

- **wide structures versus deep structures**
  Booch [1994, p. 140] discusses the subject of how to choose the inheritance relationships between classes of objects: “[Inheritance trees can be characterised as either] wide and shallow, narrow and deep, or balanced. Class structures that are wide and shallow usually represent forests of free-standing classes that can be mixed and matched. Class structures that are narrow and deep represent trees of classes that are related by a common ancestor. There are advantages and disadvantages to each approach. Forests of classes are more loosely coupled, but they may not exploit all the commonality that exists. Trees of classes exploit this commonality, so that individual classes are smaller than in forests. However, to understand a particular class, it is usually necessary to understand the meaning of all the classes it inherits from or uses. The proper shape of a class structure is highly problem-dependent.” This should also be considered for the definition of inheritance structures of Feature Types.
Similar notice should be given to aggregation structures in Complex Feature Types, although the issues to consider are different. A wide structure, i.e. a Complex Feature Type with many components, will require a great deal of management of these components at the level of that Complex Feature Type. A deep structure, i.e. a long chain of Complex Feature Types that each act as components of higher types, may ease the management of information by delegation to lower types in the hierarchy, yet the collaboration of the various components in such a deep structure is harder to establish.

- **presentation versus representation**
  Especially when defining Geometric Feature Types that represent the physical appearance of real world objects, a distinction must be made between how an instance of such a type is presented, for instance on a computer screen, and what actually comprises the concept that is represented by this type. It is the latter kind of information that must be modelled and stored in Feature Instances, for it represents the concept in the model. The presentation of the concept in whatever form is desired will be generated from the information that is stored in the representation. How this is done, in the case of Geometric Feature Types, must be defined in the description of the parametric Geometry.

- **decomposition, association, or specification**
  The choice for the kind of relationship that is to be modelled, either at the type level or at the instance level, is not always obvious. Is a door associated to the wall that it is in, or does it belong to the wall and is this a case of decomposition? This issue has been addressed in section 7.5 on the typology of relationships. Some additional relationship issues are addressed in section 8.2.

- **redundancy, completeness, and consistency**
  Although modelling a design information structure should aim at minimal redundancy and maximal completeness and consistency, it must be realised that the optimal configuration of information does also rely on aspects like reusability and practicability. Especially these two aspects will often justify certain levels of redundancy to exist in a collection of Feature Types. The pursuit for completeness should always be considered in the context and purpose for which a Feature Type is to be used and in relation with the amount of information that is likely to be available at the time of modelling or that designers are willing to provide. From the point of view of information management, consistency should always be pursued, yet in creative design, inconsistency may, to a certain degree, be acceptable. Moreover, the option to be inconsistent in dealing with information during design is often considered an important factor in creative processes.
Two distinct situations are recognised in which Feature Type definition may be initiated. One is the situation where a particular pattern of information, modelled in structures of Feature Instances, is acknowledged by the designer as representing a particular concept that will recur during the same or other design cases. The definition of a new Feature Type can then be done on the basis of the structure of Features that was modelled using relationships at the instance level. The ‘prototype’ that the designer has built by creating this structure of Features is turned into a new Complex Feature Type that defines the relationships as its components. This procedure of turning a Feature structure into a Feature Type definition could also be initiated by a design system. Using pattern matching algorithms, a design system can search for recurring patterns of Features and relationships at the instance level. Once such a recurring pattern has been found, it may be proposed to the designer as a concept of design. With reference to Feature recognition in the original area of Feature-based modelling, this process of discovering new concepts in design can be called Feature Type recognition.

The other situation in which Feature Type definition is initiated, is when a designer (or e.g. an organisation for standardisation) decides to formalise a concept that has not necessarily been modelled in terms of Features before. The formalisation of such a concept is started from scratch. For this approach, a procedure is described in the next few pages, that guides a designer through the various decisions to be made when defining a new Feature Type. This procedure leads to a selection of the appropriate class of Feature Type and assesses the definition of all its attributes, possibly resulting in the definition of other Feature Types or the instantiation of Feature Instances.

The figures 8-1 to 8-4 show the procedure for formalising a concept and defining a Feature Type. This procedure assumes that the concept has been identified by the designer, according to the previous section. It comprises four diagrams that guide the designer through a number of decisions regarding the contents of the concept.
Diagram 1 starts with the determination of the primary nature of the concept. It distinguishes procedural, geometric, and constraint concepts from all others. This distinction may lead to the definition of a Handler Feature Type, a Geometric Feature Type, or a Constraint Feature Type respectively. If none of these apply, one is to proceed with diagram 2 of the procedure.

A Handler Feature Type requires the selection of the event that is to trigger the procedure defined by the handler. The parameters for this procedure need to be declared, meaning that the types are specified of the Features that can be passed as parameters to this procedure. The procedure itself must be defined, using one of the procedural languages made available by the design system.

In case of a geometric nature of the concept, a Geometric Feature Type is defined. This type requires selection of the parametric geometry that it represents. The available kinds of parametric geometry depends on the geometry modelling engine that is integrated with the design system. The selected geometry provides the types of the parameters that must be provided by instances of the Geometric Feature Type. These parameters are given from within the context of the geometry modeller. The Geometric Feature Type further declares these parameters by providing the types of Features that can be passed as parameters.

For Constraint Feature Types, the type of constraint needs to be indicated, which depends on the availability of constraint solvers or testers in the design system. Once the constraint type is selected, the parameters required by this constraint are known and the Constraint Feature Type further declares these as the types of Features that can be passed.

After the definition of any of the above Feature Types, there may be remaining aspects of the concept that have not yet been taken into account as parameters. If this is the case, the defined Feature Type is itself contained in a larger Feature Type, a Complex Feature Type, which must be defined next. First, the definition of the current Feature Type is finished by proceeding with diagram 4 of the procedure. After that, the definition of the containing Complex Feature Type can be started in diagram 2, which takes the Feature Type that has just been defined as its first attribute.
Figure 8.1  Diagram 1 of the procedure for Feature Type definition.
Diagram 2 shows how to proceed when the concept does not lead to the definition of either Handler, Geometric, or Constraint Feature Type. First, all the attributes of the concept that represent data to be stored by Features of this type are listed. For each of these data-attributes, it is considered whether or not the attribute is relevant to the majority of occurrences of the concept. This is not necessarily a very clear decision, as the term ‘majority’ already indicates, more so because the possible occurrences of the concept may not come into view clearly at this time. Nevertheless, it should be questioned if the particular attribute really contributes to the concept’s significance, or if it is relevant only for the one occurrence of the concept that the designer has in mind. If the latter is the case, the attribute should not be defined a part of the Feature Type, but rather be modelled as a relationship at the level of the Feature Instances.

The number of data-attributes that are considered relevant for the Feature Type’s definition are counted. If this number is exactly one, then the base-type of this attribute must be determined. For attributes that represent a string, integer, real, or Boolean value, a Simple Feature Type is defined. For attributes that represent an identifier chosen from a given list of identifiers, an Enumeration Feature Type is defined. If neither of the above is the case, then the attribute itself represents a complex information structure, which must be represented by another Feature Type. Because this attribute is also the only data-attribute of the concept, it might in fact be that this attribute represents the concept itself. This is particularly true if no behaviour attributes are to be defined for this concept (see diagram 4), meaning that the concept exhibits no other characteristics than those represented by this attribute. Therefore, promoting this attribute to be regarded as the concept itself should be considered. If this is found to be the case, the procedure should be restarted, taking the notion of this attribute as the notion of the concept. Else, the procedure is continued at diagram 3, where the attribute will be the first and only attribute of a Complex Feature Type.

If the number of relevant data-attributes of the concept is greater than one, then a Complex Feature Type needs to be defined and the procedure is continued at diagram 3. This is also the case if no attributes are found relevant. This may appear to be an odd case, formalising a concept that has no characteristics, yet the mere existence of a Feature Type with a given name may be sufficient to represent a particular design concept at certain stages in the development of a design or design theory. Perhaps later the content of the concept will become more clear and attributes will be added to the Feature Type that represents it. Also, behaviour attributes are yet to be dealt with, at diagram 4, which may give more meaning to the Feature Type being defined. Concepts with no data-attributes at all are modelled as Complex Feature Type that have no Components.

Modelling Architectural Design Information by Features
2. make a list of all 'data'-attributes of the concept

consider each of the attributes at A, then continue at B

does it have sufficient relevance? no → skip this attribute, it is modelled as an instance-level relationship

yes → count this attribute, it will be modelled at the type-level

number of attributes = 1?
yes → determine the base-type of the attribute

is it a string, integer, real, or boolean? yes → define a Simple FeatureType no → define an Enumeration FeatureType

Is it an enumeration? yes → specify the enumerated identifiers no → restart at 1

Is this attribute the concept itself? yes → restart at 1, regarding this attribute as the concept no →

Figure 8-2 Diagram 2 of the procedure for Feature Type definition.
If the procedure leads to diagram 3, this means that the concept must be represented by a Complex Feature Type. All data-attributes of the concept are to be defined as components of the Complex Feature Type, which are given a role name and role type (decomposition, association, or specification). For every relevant attribute of the concept the question must be answered whether or not the attribute has a constant value for all occurrences of this concept. If this is true, the Complex Feature Type will define an Instance Component, which is formed by a relationship to a Feature Instance. Possibly, this Feature Instance needs to be created.

For attributes with a value that varies for the different occurrences of the concept, a Type Component is to be defined for the Complex Feature Type. This is a relationship to another Feature Type, which, during instantiation, results in one or more relationships to Feature Instances. Possibly, the related Feature Type does not yet exist and must be defined in a new procedure started at diagram 1. For Type Components, the cardinality, domain, and default value must be specified.

After a component has been defined for each data-attribute of the concept, the procedure is continued at diagram 4 with the definition of the concept's behaviour.
Figure 8.3 Diagram 3 of the procedure for Feature Type definition.
The fourth and last diagram of the procedure for defining a Feature Type adds behaviour to the type’s definition by means of adding event handlers. First a list is made of all the behaviour-attributes of the concept being formalised. As with the data-attributes in diagram 2, all those attributes are eliminated that bear relevance only to certain instances of the concept and are not significant to the intrinsic notion that the concept represents.

For each of the remaining behaviour-attributes, the event is specified that will trigger the particular behaviour, the event handler, of the instances of this Feature Type. Next, it must be determined if the parameters that are to be assigned to the event handler will be assigned in a similar manner for all instances of the Feature Type, or if each instance will assign the parameters in their own particular manner. If the way of assigning parameters does not vary per instance, the parameter assignment can be done at the level of the Feature Type, which results in relating a Handler Feature Instance, containing the parameter assignment, to the event handler. This Handler Feature Instance may need to be created in case it does not already exist.

In the case of per instance assignment of parameters, only the Handler Feature Type can be specified for the event handler. Again, this Handler Feature Type may need to be defined if it does not already exist. The actual parameter assignment is done during instantiation, when an instance of the specified Handler Feature Type is created.

After all the behaviour-attributes have been formalised into event handlers, the definition of the Feature Type can be concluded by specifying the domain for the instances of the type, and a default value. The kind of content of both domain and default value depends on the class of the Feature Type that has just been defined.
Strategies for Feature Type definition

4. make a list of all 'behaviour'-attributes of the concept

finally, specify the domain and default value for the newly defined FeatureType

consider each of the attributes at A; then continue at B

does it have sufficient relevance?

select the event that will trigger the behaviour and create an event handler

varying assignment of parameters?

HandlerFeatureType exist?

 Handy Featurelnstance exist?

create a new HandlerFeatureInstance and relate it to the event handler

define a new HandlerFeatureType and relate it to the event handler

relation it to the event handler

Figure 8-4 Diagram 4 of the procedure for Feature Type definition.
8.2 Relationship issues

Issues of specialisation

Regarding the types of relationships that are incorporated in the Feature modelling framework, there are a number of issues to be noted. First, specialisation of Feature Types is allowed, through the usage of the Supertype attribute of Complex Feature Types, but this can lead to single inheritance only. As has been discussed in section 2.2.3, this is a choice that is also made in other recent computer languages, such as Java, in order to avoid over-complexity that would result from multiple inheritance. The advantages of multiple inheritance are considered not worthwhile.

The issues of overriding and hiding inherited components in a Complex Feature Type has not been addressed specifically in the definitions in chapter 7. No restriction has been posed on the naming of roles of components in a Complex Feature Type, which implies that reusing names of inherited components is allowed. Whether this should be regarded as overriding or hiding, is not determined at this point. Overriding would mean that the components of the subtype, when given the same name, would replace those of the supertype. If the components of the supertype could still somehow be accessed, then this would be a case of hiding the supertype's components. The latter approach is more rich than the former, and therefore preferable, but no provisions have been included in the framework as yet to access the hidden components.

Role names of relationships and standardisation

The types of relationships in the FBM framework are limited to four (excluding the Instantiation relationship between Feature Types and Feature Instances). This is a limited number because further classification of the relationships at the meta-level of the framework would limit the flexibility of the framework at this point. More detailed classes of relationships would require classification of the design domain concerning its content. This is not to say that classification of the domain of Architectural design is not a necessity, however, similar to the definition of the Generic Feature Types, which must be considered a task of standardisation bodies, the detailed classification of relationships is a standardisation task. It is recognised in this research project, as a result of the work on prototyping and case study as described in sections 8.8 and 8.9, that the function of role names in the model is important for the ability of both system and designer to interpret the model and reason with its contents. Without knowledge of the role names of relationships, the design system can only determine which of the four types of relationships has been chosen to model a particular organisation of Features and, of course, this limits the possibilities of reasoning with this knowledge. Further prototyping in the development of design systems will have to demonstrate whether these limited possibilities are sufficient for the level of reasoning that is required for design systems, or that knowledge and usage of standardised role names are essential.
Martin and Odell have studied the transitivity of relationships, which is a characteristic of relationships that allows to infer new relationships from a given set of relationships. Their conclusion is that the transitivity of relationships depends on the kind of relationships involved and is mainly limited to cases where the relationships are of the same kind: if the shoes are in the box and the box is in the bag, then the shoes are in the bag. If the kinds of relationships are not the same, the validity of the transitivity must be further examined per case. Yet, sameness of kind of relationship is not a guarantee for transitivity; it typically does not hold for relationships based on membership. If the relationship is transitive, operations can be propagated through the relationship: if the pistons are part of the engine and the engine is part of the car, then the pistons can be rotated when the car is being rotated.

From the study by Martin and Odell the conclusion can be drawn that design systems cannot simply rely on the typology of relationships when determining the validity of, for instance, propagation of operations through hierarchies of objects. This, again, necessitates the prototyping of design systems, in order to answer the question whether or not role names of relationships are necessary for this kind of knowledge inference from a Feature model.

8.3 Instantiation of Features

Instantiation of Features is the process of creating instances of Feature Types that represent the information concerning an actual design case. Primarily, instantiation of a Feature Type means that the design system and the designer together must perform the following tasks:

- selection of a Feature Type that adequately represents the concept that must be modelled in the design situation;
- creation of a Feature Instance based on the definition of the selected Feature Type;
- provision of values for the attributes of the new Feature Instance: for instance, Simple Feature Instances require input for their basic value, e.g. a string or real. In the case of Complex Feature Instances, the attributes of the instance are components, references to other Feature Instances. These other Feature Instances may or may not be already existent in the model.

---

1 Transitivity of relationships means that, if A relates to B and B relates to C, then A relates to C in the same manner. For example [Martin and Odell 1995]:

premise 1: Socrates is a man;
premise 2: Men are mortal;
conclusion: Socrates is mortal.
8.3.1 Scenarios for Feature instantiation

The three tasks of the instantiation process can be performed in a number of scenarios, each characterised by different circumstances in which information is getting defined and by different approaches for handling this information. This subsection describes some of these scenarios.

**Straight-forward instantiation by the designer**

The first scenario is the most obvious one, at least from the point of view of system-developer. The designer decides to model a piece of information by selecting the appropriate Feature Type from a library, generating an instance of this type, and providing it with the information available from the design. Depending on the content of the Feature Type definition, providing the information can be quite a task. This is aggravated when the Feature Type is a Complex Feature Type of which the components are Feature Instances that are not yet available in the model.

**Working with defaults**

Obviously, design information must somehow be entered into the model, but not necessarily all at once, and not necessarily by the designer. The system should allow the designer to postpone entering information if that information is not available or not considered relevant by the designer at that moment. One way of managing this laborious task is to make use of default values. The framework allows each Feature Type to define a default value for its instantiation and also the instantiation of components of Complex Feature Types can be supported by accepting the defaults for these components. In this manner, providing the actual information is postponed while the model is still relatively complete. Moreover, changing existing information in a model is often experienced to be less tedious then having to provide a lot of information from scratch. For instance, modifying the dimensions of a building element that is already displayed on screen is much easier than having to enter all the details numerically.

**Handling incompleteness**

Working with default values for Features partly solves the problem of how to deal with situations where a designer cannot or does not want to provide the information necessary for complete instantiation of a Feature. Yet accepting these defaults is not necessarily what the designer has in mind when developing the model; it may seem more appropriate to the designer to leave certain details undefined, rather than to have unconfirmed values entered in the model. This situation can occur, for instance, in cases where the value of a Simple Feature cannot yet be given, or the number of components of a Complex Feature required by its cardinality cannot yet be provided. A design system must allow incomplete instantiation of Features, where the value of Simple Features and the relationships of Complex Features are undetermined. During the process of design and modelling, the system should allow this incompleteness and the possible
inconsistencies to exist, even if this should lead to undetermined and ambiguous situations in the model. Many designers will regard this 'unstructured' way of working an essential aspect of creative design. Consequently, facilities for checking the completeness and finding omissions in the model are important requirements of the system.

**Handled instantiation**

One of the events aforementioned at the introduction of event handlers is the instantiation of a Feature. Event handlers can be used to assist in providing values for the Feature attributes and in the creation of other Feature Instances that are to be related to the newly instantiated Feature.

An example is given to clarify this, which assumes the definition of a Feature Type called Space, with attributes for its Height and the Level of the building on which the space is located.

```plaintext
complex Level {
    Spec Height height;
    ...
}

handler CreateSpace {
    Procedure {
        // this procedure relates the attribute NOTIFIER.height
        // to the attribute NOTIFIER.storey.height
    }
}

complex Space {
    Spec Height height;
    Assoc Level storey;
    Behaviour { CreateSpace{onCreate} create; }
    ...
}
```

The event of instantiating a new Space Feature may be handled by that Feature by automatically relating the attribute height to the height of the Level Feature where the space is located (found in the attribute storey). This results in a structure of Features where the height of the Level is shared with the height of the Space. Consequently, the designer needs not be asked to enter a value for the height of the space, moreover, future changes to the height of the level are immediate changes to all spaces at that level.

**Macro instantiation and Feature templates**

Some major disadvantages of using handlers for automatically assigning values to Feature relationships must be noted. The first is that only one method can be specified to handle the event of instantiation. Obviously, the executed procedure may include various scenarios, based on further designer input, yet these must all be included in the code embedded in the Handler Feature Type. Secondly, it is not the designer who takes the initiative to instantiate the Feature in the manner prescribed by the handler. Again, the
Implementation of the FBM Framework in Architectural Design

designer can be consulted by the procedure whether the action should be continued or not, but this can hardly be called initiative. A third disadvantage is that the instantiation process is started only once a Feature Type has been selected for instantiation and therefore depends on the availability of information to make that selection. This process is related to a specific Feature Type, which very much limits it to a particular sequence of designing. For example, to relate the height of a space to the height of the level that contains the space, first the level must be instantiated, then the space.

The above objections to the usage of handlers for automated instantiation procedures lead to the conclusion that such a procedure should not be embedded in the Feature Type. Instead, the system should allow procedures to be defined, stored, and executed independently from the Feature Type libraries, similar to the way most CAD systems allow the execution of commands to create geometry. Since the content of the procedure to be executed is a responsibility external to the system, they cannot be embedded in the system, but must be specified in some form of programming language. Generally, this kind of customised behaviour of computer software is called macro execution, or scripting. The scripting language must provide sufficient access to both Feature Type libraries and Feature models, in addition to general facilities for interfacing the procedure with the designer and the modelling system.

Another approach to solve this problem is the usage of prepared structures of Feature Instances that represent regularly used configurations. This approach differs from the usage of default values or handlers, in that these prepared structures, or Feature templates, can co-exist with the default values and handlers, since they are defined outside the scope of the Feature Types. This allows different configurations for different situations to be defined, which is desirable for the needs of the individual designer.

Feature recognition

An important issue in the original area of Feature modelling in Mechanical Engineering is Feature recognition. In that area, Feature recognition has the meaning of recognising Features from a given geometric model. The geometry is analysed and searched for patterns of geometry that match the definition of known Feature Types. Once such a match is found, the geometry can be replaced by an instance of the found Feature Type. In this manner the geometric model, which is poor in semantics, is converted to a Feature model that provides all the additional information necessary to, for instance, manufacture the geometry of the designed product with the available machinery.

Architectural design systems may well benefit from a similar approach to designing elements of a building. Providing the designer with generic geometric modelling tools, the created geometry may be analysed and interpreted as a structure of Features that semantically enrich the geometry with detailed architectural information. This approach is, of course, limited to those Features that can actually be discriminated on the basis of their geometric representation. Using inference methods, these geometrically recognised Feature structures may eventually be enhanced with additional Features that are defined as relationships to the geometric Features. For example, once a wall Feature has been recognised from the geometry created by the designer, Features
such as material, construction method, cost, maximum load, etc. may be inferred from the existence of the wall Feature and added to the list of relationships of that Feature.

Another kind of Feature recognition that can assist the designer in building up a consistent and semantically rich design model, is to try and recognise patterns of Features not from a geometric model but from the Feature model as it is being created. Here, it is not the bare geometry that is matched to definitions of Feature Types. Instead, in the Feature model the instance level relationships between Feature Instances are analysed and compared to the structures of Feature Types in available libraries. In this manner, a given constellation of Features that are interrelated by the designer during modelling at the instance level, can be replaced by an instance of a Feature Type that has been found to define the same relationships at the type-level. This facility of the design system supports the designer in creating consistent models and adding knowledge to the model that is implied by the design actions. The degree of similarity between found Feature structures and the relationships in a particular Feature Type should possibly be variant, allowing the designer some freedom in using accustomed terminology and including cases that are similar to defined Feature Types. Mainly the latter may well appear to be a stimulant to the designer, since the system is now encouraging the creativity of the designer and helping the development of the design as it proceeds. In fact, this form of Feature recognition could well support the implementation of case-based design as discussed at the beginning of this thesis in section 1.4.3.

8.4 Dynamic manipulation of Feature Types and Feature models

The FBM framework presented in this work allows designers to define Feature Types and to use them subsequently for modelling design information. An important problem with this flexible approach is, how to deal with modifications of Feature Types after their initial definition. With the facilities that are offered by this framework, it is very likely that designers will keep improving the definition of their Feature Types as they continue designing. Without special provisions in the design system, this will lead to unacceptable inconsistencies in both Feature Type libraries and Feature models. The design system should therefore react to any modification of a Feature Type's definition by:

- Analysis of the consequences of the modification for the definition of the Feature Type itself. This involves checking the consistency of the definition.
  Example 1: if the domain of a simple Feature Type is modified, check if the default value is still within this domain.
  Example 2: if a component of a Complex Feature Type is modified, validate all the references to that component from within the other parts of the Feature Type, such as the event handlers.
  In case of inconsistencies, the system should attempt to correct the situation and, in ambiguous cases, consult the designer for the measures to take. Automatic correction is possible, for instance, when the modification involves renaming...
roles of components, or even modifying the Feature Type of a component, as long as the new definition is compatible with the previous situation. Replacing the Feature Type of a component with a Feature Type that inherits from the former type is an example of compatible modification that can be propagated throughout the definition automatically.

- Analysis of the consequences for other Feature Type definitions. Similar to the analysis of the definition of the Feature Type itself, external references to the modified parts of the Feature Type must be validated and possibly corrected. This is potentially a very laborious task, depending on the number and size of the Feature Type libraries that must be analysed and on how generic the modified Feature Type is; in other words, how often it is referenced by other types.

- Analysis of existing Feature Instances, validating and correcting the references and values in these instances. This includes Feature Instances in Feature Type libraries as well as in Feature models. Validating Feature Instances is conceivably an even more tedious process, depending on how many instances have already been created of the particular Feature Type.

Clearly, modification of a Feature Type's definition may well have drastic implications, that can result in time consuming correction activities for the design system or the designer, but probably for both. This appeals for extra functionality of the design system to either prevent, postpone, or avoid these laborious processes. Prevention involves warning the designer for the possible consequences of the modifications that are being made to a Feature Type, for instance by reporting the number of references and instances that are potentially invalidated by the modification. This warning should be accompanied by the option to follow other approaches for defining the desired Feature Type, such as creating subtypes or copies of the Feature Type in question.

The laborious correction processes can also be postponed, which is to say that the analysis of references and instances is not done immediately. This implies that the possible inconsistencies that result from Feature Type modifications are tolerated and resolved only at a later stage, when problems are encountered or when the design has arrived at more final stages where consistency becomes more important. The choice to follow this approach should be the designer's. Design systems can support this approach by automatically checking the validity of both Feature Types and Instances that are accessed during the design process, and by providing a procedure for overall consistency checking of both Feature Type libraries and Feature models.

Finally, the problem of inconsistency in Feature Types and Feature Instances as a result of Feature Type modification can also be avoided altogether by administering version information of Feature Type definitions. Version information is currently not included in the definitions of the various classes of Feature Types and Instances in the framework. However, version control is becoming an important issue in commercial Object Oriented Database Management Systems. Issues in version control are:
• static references (to a specific version) and dynamic references (selected version depends on environment);
• multiple levels of versions: for representations, alternatives, versions;
• change propagation by means of either message-based or flag-based notification.

Naturally, including version information in the Feature Type definition does not solve the inconsistency problem that is caused by inconsistent modelling within one version. This kind of inconsistency can only be approached by evaluation of the information structure defined by the Feature Types, which most likely requires knowledge of the semantics of the formalised concepts.

8.5 Standardisation and communication

An issue of importance for the implementation of the Feature modelling approach in architectural design is how this approach relates to standardisation efforts in the area of information modelling for AEC in general and B&C in particular. Currently there are two main developments in this direction, STEP and the IFCs, which have already been briefly introduced in chapter 2.

In comparison to STEP

In the Feature modelling framework, it is the Generic Feature Types that are to be standardised for the domain of architectural design and for the Building & Construction industry in general. In relation with STEP, the Generic Feature Types take a position in between the Application Protocols and the Integrated Resources. Two issues must be considered regarding this relation. First, the FBM framework is developed for purposes of flexibility and extensibility. Whereas the Application Protocols form rigid conceptual models at the basis of the implementation of computer applications, the Generic Feature Types form only a part of the conceptual data model that can be extended with Specific Feature Types, even at the user level of computer applications. However, standardisation, and with it the development of generic data exchange methods, must be based on the availability of the Generic Feature Types.

Second, the Generic Feature Types should ideally be defined such that the chains of relationships between the various types are as short as possible. Their definition must be a balance of sufficient semantics and a minimum of interdependencies between types. This is because smaller structures of data definitions are more versatile, allowing the application thereof in various domains and thus facilitating easy exchange of data models.

Libraries of Generic Feature Types that are defined in this manner take a position, in terms of semantics and data-exchange capabilities, between the IRs and the APs. They potentially form a semantically richer basis for data exchange than the IRs, while, at the same time, avoiding the rigidity of APs, which makes them more versatile.
A similar position is to be taken by the Building & Construction Core Model (BCCM) [ISO TC184 1996], which is a development that aims at the definition of an AP that contains information structures common to the B&C industry. The BCCM is proposed to function as a common basis for B&C APs, instead of the semantically poor basis formed by the IRs. As such, it should enhance the interoperability of the B&C APs, providing them with domain specific, shared content.

**In comparison to the IFCs**

The purpose of the IFC development is to provide the CAD vendors and third party developers with a means to create computer applications that can collaborate in the whole area of design in the B&C industry. This leads to very different requirements for the model’s architecture than those of the FBM framework, because flexibility and extensibility at the user level of data definition are not an issue. However, parallels can be drawn if the Generic Feature Types are placed at the level of the resources layer and the core layer of the IFC architecture (see figure 2-9 on page 42). This would position the Specific Feature Types at the top layer of IFC domain models. The connecting interoperability layer in the IFC development is achieved using formal mapping languages. This technique of translating one conceptual model to another will also be necessary for the communication of Specific Feature Types from different sources, using the Generic Feature Types as interface between them.

**Mapping algorithms**

The FBM framework in this research project has been initially developed for the benefit in early design stages by facilitating the dynamic characteristics of early design. Communicating dynamic, non-predefined data structures to other participants is likely to be a very complicated matter. However, expectations are that the development and standardisation of libraries of Generic Feature Types can be a bridge in the issue of data exchange. The main problem with the exchange of Feature models is twofold: how can instances of Specific Feature Types be exchanged, and how are non-typical relationships interpreted by other parties?

Exchange of instances of Specific Feature Types will require mapping algorithms [Amor and Hosking 1994], such as those being developed for the mapping of APs in STEP [ISO TC184 1994b] and similar to the interoperability layer of the IFC architecture. According to a survey and comparison of mapping methods used in product modelling areas, by Liebich et al. [1995] (see figure 8-5), mapping methods may involve:

- equation of data entities
- collapse of structures of data entities
- construction of structures of data entities
- unit conversions
- type conversions and pointer conversions
- splitting or combining collections of data entities
In order to do so, mapping algorithms require the availability of:

- entity definitions on both sides of the mapping
- the definition of conditions for a mapping to take place, including a mechanism to discriminate between the conditions
- default values for attributes of entities to be created
- information on the cardinality of the mapping
- a mechanism for referencing values through chains of pointers
- methods of iteration over collections of data entities and their attributes.

Mapping algorithms in the FBM framework can be based on the definitions of Generic Feature Types, using these standardised types as intermediary model for the mapping between different collections of Specific Feature Types.

Information that is not defined at the type level in the FBM framework, is represented by the instance level relationships in a Feature model. The exchange of this kind of information is much harder to realise than the mapping of types. Since a minimum of knowledge about the relationships must be available to mapping algorithms, it seems that the usage of these instance level relationships must somehow be structured in a consistent and formal manner. In terms of the FBM framework, this means that the role names of these instance level relationships cannot be chosen randomly but must follow certain conventions, that is, if automatic mapping of the model is an issue. Standardisation of these role names, in addition to the standardisation of the Generic Feature Types, then becomes a necessity, as has also been discussed in another context in section 8.2.

Evaluating the requirements for mapping methods as summarised above, it is expected that an implementation of these algorithms can be done using the Handler Feature Types of the framework. Although the language for the specification of the procedures of the Handler types has not yet been defined in this project, the requirements for this language, as described in section 7.7.2 on page 151, provide a
ground for the presumption that its functionality will match the requirements of the mapping algorithms.

8.6 Interface-aspects: a metaphor for Features and Feature Types

The following two sections in this chapter deal with issues of implementation of the FBM framework in a design system that is currently being developed in what is called the VR-DIS\(^2\) research programme of the faculty of Architecture at Eindhoven University of Technology. The VR-DIS programme itself is introduced in the introduction to this thesis. This section highlights one of the tasks in this programme, namely the development of the user interface of the design system.

Many Features in an architectural design model will represent concepts that have a physical appearance in the design; they represent building elements such as walls and floors. Other Features can be easily associated with a physical appearance, either because they represent a spatial, though invisible, concept such as a space or a route, or because they represent a collection of other Features that have a physical appearance. An example of the latter is the concept ‘elevation’ which by itself is not a single building element that has physical appearance, but rather represents the collection of building elements that belong to what is known as the elevation.

Probably the majority of Features in any model of architectural design represents concepts that are not visible in the real world. They cannot directly be related to objects that are found in the eventual building, either because they represent physical aspects that are invisible, e.g. a material’s R-value, or because they represent abstract, non-physical aspects, such as costs or function. The interface to a computer supported design system must, however, present all kinds of Features to the human user and allow the user to interact with the model on the basis of these presentations.

The problem of user interfaces to abstract data models is addressed in the VR-DIS programme by Coomans [1998; Coomans and Timmermans 1998]. This section briefly describes the current status of a part of this development that deals with the visualisation of abstract data in Feature models. In his research project, Coomans departs from the point of view that using experiential skills of the user will make a design system much easier and intuitive to use. The most effective experiential skills that humans acquire are the perception capabilities, especially the perception of spatially structured visual information, the human motor capabilities, and the human language capabilities. The perception and motor capabilities are based on our experience of the world around us and is therefore based on three dimensions. Of the computer technology developed so far, Virtual Reality (VR) technology is best capable of exploiting these human skills of

---

\(^2\) VR-DIS is a dual acronym: from a technical point of view it means Virtual Reality – Distributed Interactive Simulation; from the point of view of application, it means Virtual Reality – Design Information System. The combination of both meanings covers the contents of this research programme.
dealing with 3D perceptions and actions. The combination of 3D computer imaging and 3D motor control of computer representations can potentially reduce the effort needed to control a design system to the same level as is needed to control our everyday environment. Among the most important factors for the success or failure of this interaction paradigm is what is usually called the interaction metaphor: the relation between the system's semantics and the user's actions. In other words: the presentation of the computer model to the user and the computer-tools available to the user for manipulation of the model determine whether the user will be able and like to work with the design system or not.

Focussing on this fact, Coomans has designed an interaction paradigm in a Virtual Reality environment that allows the design system to provide the user with both a pictorial presentation and a verbal presentation of the architectural model (see figure 8-6). The pictorial presentation is used for the kinds of Features that can be made visible in a way that resembles what we know from the real world. In VR this presentation will manifest a high level of veracity, not only because of the quality of the images, but also because of the interaction it allows the user to have with the model in three dimensions. This pictorial presentation is used not only for those Features that have their visible counterparts in the real world, objects and their visible characteristics, but also Features that can be made visible using characteristics of the real world, such as the colour of objects in the model. An example of the latter is the visualisation of heat loss of building elements by varying their colour from blue to red as an indication for the amount of heat that is lost.

The verbal presentation is required for the kinds of Features that cannot directly be related to things that are visible in the real world: for these Features, a visualisation metaphor must be used in order to allow the user to interact with them. A graph-like
metaphor is chosen for this presentation of the Feature model, where the Features are presented as the nodes in the graph, and their relationships as the arcs in the graph.

More precisely, the Features use the metaphor of a three dimensional box, with information concerning the Feature written on the front of this box. The relationships between Features are shown using ties between these boxes, with information such as role name written on the ties, while the location of the connection of the ties to the boxes indicates the kind of relationship: decomposition, association, or specification. Boxes can be opened and closed, to reveal their relationships. When a box representing one Feature is opened, the related Features appear as smaller boxes surrounding the opened box. The focus of the user, observing the boxes in the virtual environment, can change from one Feature to another, in which case the size and position of the boxes change. The boxes are organised in a three dimensional grid that takes the form of faceted spherical shells that are transparent and presented towards the user as is shown in figure 8-7. New shells are created to display more Features and laid concentrically in front of the other shells. In this manner, the history of browsing through the model remains visible through the transparent shells. When focus shifts from the current Feature box to a related Feature box which is shown smaller, this small box first moves to another facet of the shell, then increases size and opens, revealing its own relationships. If necessary, a new shell is created and laid in front of the current shell, which moves backwards. The ties between the boxes, representing the Feature relationships, are flexible and maintained as boxes move position across the shells.

Other issues of interest in this research project deal with the development and application of input devices and their metaphors in VR. Examples of these are a three dimensional mouse, a trackball (which, in this project, are preferred over a so-called spaceball), and a turntable, each with their own characteristic way of controlling movement, position, and orientation of objects and of the observer in the virtual environment. For example, the 3D mouse is connected with a virtual pen in the

Figure 8-7 Interface design for Feature representation in a Virtual Environment. (from [Coomans 1998])
environment of the model, that can be used to grab objects and manipulate them in 3D space.

Output of the design system is another issue in the research project of Coomans. Based on the conclusions from previous research that the 'traditional' immersive virtual reality environment, using e.g. a head mounted display or all-side projection configuration, is not ideal for longer sessions of design tasks, a desktop virtual environment is now being developed based on what is called *fish tank VR* [Ware et al. 1993]. This virtual environment is viewed on a monitor with the perspective projection of real time generated images that are coupled to the head position of the observer. The images can be either stereoscopic or monoscopic. With this technique, the video screen is experienced as if it were the glass front of a fish tank, in which the virtual objects are contained. The perspective projection of these objects is adjusted in accordance with the observer’s head movements, such that the correct perspective to the objects is obtained from any viewing angle.

### 8.7 VR-DIS development: system architecture and implementation issues

The VR-DIS research programme, as introduced on page 3, aims at the development of an interdisciplinary design system that uses VR technology as the main user interface. The programme’s development takes place in phases of which the first phase aims at early design stages. Since the FBM framework, presented in this thesis, is developed for the support of architectural design, this theory has been adopted as the approach for modelling design information in the VR-DIS research programme.

**Modular architecture of VR-DIS**

The choice for the FBM framework as the modelling basis for the design system, together with the choice for a VR environment as the principal user-interface, has led to
the proposal of a modular architecture for the design system, which is shown in figure 8-9.

This architecture comprises four modules initially, but can be expanded in later stages, as the diagram suggests. At the basis stands the Feature Management Application Programming Interface (API). This module is responsible for all access to Feature related data, which includes both Feature Type Libraries and Feature models, providing the functionality needed to define, store, retrieve, and manage Feature Types and Instances. The other three modules are developed using this API and their functioning depends on it.

The Geometry Engine (GE) is the module that generates and manages geometric data. This data is not necessary to define design information, but is merely required to visualise geometry that is defined by the parameters provided by the Feature data. The parameter retrieval is done through the API. Geometry that is generated on the basis of Feature data is communicated to the next module, the Virtual Reality User Interface.

The Virtual Reality User Interface (VRUI) module visualises the geometry in the VR environment, together with other parts of the interface to the Feature data. The VRUI is the sole user interface to the design system. All control of design data is done

![Modular architecture of VR-DIS](image-url)

*Figure 8-9 Modular architecture of VR-DIS: The various modules can read-access the Feature data through the API, but other access must go through the Feature Manager.*
through this interface, for which it has retrieval access to the Feature data through the API. A great deal of the control of design data is related to the geometry of the design, which is realised by the usage of control-points for the geometry, positioned for example at the vertices and middle point of edges and at the centre of faces. The geometry of control-points is again managed by the GE module, but the interaction is controlled by the VRUI. Other parts of the interface involve the visualisation of abstract Feature data, by means of the 3D metaphors as is discussed in the previous section.

The central module is the Feature Manager. Whereas both the GE and VRUI modules can retrieve Feature data directly through the API, the Feature Manager module is responsible for the creation and maintenance of Feature data. Apart from the retrieval of Feature data, all access to Feature data that modifies it is done through the Feature Manager. Its main tasks are to maintain consistency of the Feature data and integrity of the models as multiple users are accessing the data in various manners.

Figure 8-9 shows two more boxes representing modules of the system: the Constraint Solver module and a box indicating that other modules can be added to the system in later stages of the development. The Constraint Solver is a module that is expected to be added in the near future; integration in the prototypes of VR-DIS of object oriented constraint solving techniques developed by Kelleners [1999] has already delivered good results and predicts that constraint solving will be a valuable addition to a system supporting creative design.

8.8 Prototyping the Feature framework in the Feature Management API

In the proposed system architecture of the VR-DIS developments, the Feature Management API forms the implementation of the FBM framework that is presented in this thesis. As such, prototyping of the API forms a test case for the theory of the framework. The prototyping of this module has been planned in as many as four stages, of which the first three have been completed at the moment this is written. Division of the prototyping efforts into phases has been enforced by the complexity of the framework and the implementation that it requires.

In general, the development of the prototypes departs mainly from the following assumptions and requirements:

1. The system is based on the Object Orientation paradigm. The usage of a readily available object oriented data management toolkit is preferable.
2. The architecture of the API is object oriented, resulting in a collection of object classes that provide a programming interface to the functionality of the framework, but hide the underlying data management tasks.
3. The implementation of the object-classes reflects as close as possible the FBM framework as designed.
4. The eventual API will be embeddable in different hosting languages and development environments, running under the Windows NT platform, such as
8. The design of the API, if not its implementation, is prepared for (a) multi-user access to the Feature data, and (b) distributed implementation of the design system.

8.8.1 Phases of the prototype

The Feature Management API is prototyped in four phases. The first two phases of this development can truly be indicated as prototypes, since their purpose was to test the design of the framework. The latter two phases are no longer prototypes in this sense, but form the start of a fully functional API. However, in the overall development of the VR-DIS design system, they still form a part of the prototyping of this system.

8.8.2 Phase 1: Feasibility test

The first phase concentrated on the first three issues in the list above. Its goal was to investigate the technical feasibility of an implementation of the FBM framework. Hence this first prototype has been developed parallel to a very early stage of the development of the framework. Its development involved these principal steps:

- selection of an OO data management toolkit;
- design of the object-classes in this toolkit for the representation of the different classes of Feature Types;
- development of an elementary testbed for assessment of the functionality of the prototype.

The scope of this prototype was limited to implementation of classes of Feature Types only. Moreover, not all classes of Feature Types were involved in the prototype, only the classes for Simple Feature Types, Enumeration Feature Types, and Complex Feature Types were implemented. This was considered sufficient to answer the primary purpose of the prototype, which was to demonstrate the technical feasibility of a system allowing designers to formalise design concepts into Feature Types.

Two important conclusions could be drawn from this prototyping phase, the first being that implementation of the FBM framework, as far as it concerned the functionality of user-defined Feature Types, was technically feasible. The second conclusion was that full implementation of the framework would require thorough software engineering and could not be planned in short term.

8.8.3 Phase 2: Functional assessment

The second phase of prototyping served a purpose which could not be served with the prototype from the first phase. This involved the testing of the applicability of the theory in the FBM framework with respect to the activities of architectural designers. Since the first phase's prototype had only implemented half of the framework (the part of the

Modelling Architectural Design Information by Features
Feature Instances was missing) and because that prototype lacked a user interface, the objectives of the second prototype were formulated as follows:

- rapid development of a rudimentary, though practical, interface to the functionality of the framework;
- implementation of both activities of definition of new Feature Types and creation of Feature Instances.

These objectives, and the experiences from the first phase of prototyping, urged the decision to re-interpret the functionality of the FBM framework and to develop a minimal approach for implementation of the representation of Feature data for this prototype. This led to the design of a relational database for the storage of the Feature Type definitions and Feature Instances. Again, the implementation was also restricted to classes of Simple, Enumeration, and Complex Feature Types only and their respective instances.

The result of this second phase was an application with a Windows-based user-interface that allowed the definition of the aforementioned classes of Feature Types and their instantiation. An impression of this application, with the working name of Feature Tool, is shown in figures 8-10 and 8-11.

![Figure 8-10 Interface of the Feature Tool, the second prototype, for defining Feature Types.](image-url)
This Feature Tool was used in an analysis of a case of an actual architectural design and in the implementation of a design system prototype. The case is discussed in section 8.9. The design system prototype formed one of the first developments in the VR-DIS context. This experiment involved the prototyping of a design system that was based on the conclusions from previous experiments and that integrated other modules developed in the VR-DIS programme. The resulting system consists of a module for geometry modelling that uses a virtual reality interface, a module for modelling and solving geometric constraints, and the Feature Tool prototype. It is aimed at the early design stages, allowing the modelling of the morphology and topology of spaces and structures of a building. Furthermore, the system allows the assignment of constraints to these elements of the building, which assists the designer in meeting the design brief. Such constraints involved relationships between elements such as adjacency and alignment. Details about this experiment are reported by de Vries and Jessurun [1998a, 1998b].

8.8.4 Phase 3: Limited API development

The third phase of prototyping really forms the first step in the actual implementation of the Feature Management API. This implementation builds on the results of the previous two prototypes, adopting the OO approach from the first prototype and using the experiences with the second prototype regarding the required functionality of a Feature Management tool. Yet, the architecture for this third implementation has been thoroughly revised, taking into account the complete set of requirements that have been introduced at the beginning of this section.
The Feature Management API consists of a set of object-classes implemented using DCOM\(^3\) technology. These object-classes disclose the functionality of creation and manipulation of Feature data, including Feature Type Libraries and Feature models, and perform the full management of this data in an object oriented database. This database was implemented using ObjectStore PSE\(^4\). The current implementation of the API allows these object-classes to be used in any programming environment that supports the usage of DCOM. Embedding the API objects in an application allows the creation of Feature Type Libraries and Feature models that are stored either locally for usage only by the computer running the application, or central in a computer network for distributed access to the same Feature data.

The API object-classes are used to implement the next generation VR-DIS prototypes. These include an application with a Windows-based interface to a Feature Management application, similar to the Feature Tool developed in phase 2, and a prototype for accessing Features from within a VR environment, using the interfacing metaphors as designed by Coomans (see section 8.6).

8.8.5 Phase 4: Full API development

Phase 3 of the prototype, the first version of the API, is limited to an implementation of Simple, Enumeration, and Complex Feature Types and Instances. Phase 4 will consist of the expansion of the implementation with the other classes from the FBM framework. It will also include:

- an implementation of the language for writing event handler procedures, and
- facilities for consistency management of the libraries and models.

8.9 The application of Features in an architectural design case

The theory of the FBM framework is used to describe an actual design case. In this way, it is possible to assess the applicability of the framework in a design context. This case study uses the Windows-based Feature Tool from the prototyping phase 2, described in section 8.8.3. The study was performed by an experienced design researcher from the Design Systems group at Eindhoven University of Technology who analysed the work of an actual design that was executed by an architect’s office. To allow for a more objective

---

\(^3\) DCOM: Distributed Component Object Model is a Windows-based programming technique that is based on facilities provided by Windows to share software-modules in multiple applications, across a network of computers. This technique uses a standardised interface for accessing these software-modules that follows an OO approach.

\(^4\) ObjectStore PSE is the single user Personal Storage Engine version of ObjectStore providing a software development toolkit for object oriented database management by Object Design (www.odi.com).
evaluation of the modelling approach, the author of this thesis was not involved in the actual definition of the Feature Types and the modelling in Feature Instances.

The case study is a design of a single family house, which is analysed in terms of Features [Achten and van Leeuwen 1998]. In the architect’s office, AutoCAD was used from the very start of the design work. Drawings during the design process were made as new copies rather than changing or revising old drawings. In this way, key phases of the design process are available for analysis.

The case study is based on 30 drawings made during the design process. Each single drawing is taken as a step in the design process for which a Feature model can be established. The transition from one step to the next therefore represents the design decisions taken from that phase to the next. Figure 8-12 represents the first six phases in the design case.

- Definition of a phase: one single drawing. Notation: phase \( n, n \in \mathbb{N} \).
- Definition of a step: the transition from phase \( n \) to phase \( n+1 \).
- Definition of the design brief as phase \( 0 \).

The analysis of the design case is carried out according to the schemes in figures

![Figure 8-12 Drawings from the first six phases of the design case study [Achten and van Leeuwen 1998].](image-url)
8.9.1 Example of Feature modelling in the case

The brief of phase 0 provides a lexicon of design elements that play a role in the building design. One example is the concept of space. Spaces such as hall, toilet, and wardrobe are elements of the lexicon. First, Feature Types have to be defined, after which Feature Instances can be made.

**Feature Types**

A Feature Type called *Space* is defined of which the various spaces in the brief are instances. The text in the brief notes the following aspects of a space:

- function (such as: bedroom and bathroom);
- contained elements (such as: stair and toilet);
- visual relationship (such as: kitchen closed with respect to living);
- access relationship (such as: doors to garden and bathroom);
- daylighting (such as: daylighting in kitchen);
- adjacency (such as: scullery between garage, kitchen and bathroom);
- roof type (such as: no glass roof);
- number of persons (such as: guest room for two persons, dining room for 8 persons).

From this summary, it is clear that *space* is not a procedural, geometric, or constraint concept. Therefore, it has to be a complex Feature Type (see figure 8-1). Determining which aspects are to be included in the definition of the Feature Type *Space* and which aspects are to be defined in other Feature Types is not straightforward (see figure 8-2). If the aspect concerns only the space itself and does not strongly refer to other elements, then it can be included in the Type. Following this guideline, constraint-like relations such as visual relationship are better defined independent of the Space Feature Type. Function, contained elements, daylighting, roof type, and number of persons are part of any space and therefore are included in the type definition. The Feature Type *Space* is defined accordingly and results in the following:
Implementation of the FBM Framework in Architectural Design

complex Elements.spatial.Space {
    TypeDescr("Space element where activities can take place")
    TypeAuthor("Henri Achten")
    Has Elements.spatial.Space contains[0..?];
    Spec Arch.value.Daylighting daylightUsed;
    Assoc Elements.spatial.Function function;
    Spec General.value.Number numberOfPersons;
    Spec Geometry.dim.Height height;
}

In order to complete the Feature Type definition of space in this phase, the Feature Types Daylighting, Function, Number, and Height must be defined as well.

boolean Arch.value.Daylighting {
    TypeDescr("Indicates daylight as a lightsource")
    TypeDefault(true)
}

string Elements.spatial.Function {
    TypeDescr("Main activity in a space")
}

integer General.value.Number {
    TypeDescr("An integer number")
    TypeDefault(1)
}

real Geometry.dim.Height {
    TypeDescr("Height in m")
    TypeDefault(3)
    TypeUnit("m")
}

Figure 8-13 Graphical notation of the definition of the Feature Type Space in phase 0 of the case study.

Figure 8-14 Definition of the Feature Type Space_adj_Space

The Feature Type Space is not only used to make Feature Instances of spaces such as hall and kitchen, but also in all other cases where in the brief of phase 0 a space is mentioned. This is the case, for example, in the adjacency relationship between two spaces. The corresponding Feature Type is Space_adj_Space:

complex constraints.spatial.Space_adj_Space {
    Assoc Elements.spatial.Space access[2..2];
}
Feature Instances

The Feature Types are instantiated into the particular Feature Instances used in the design. In the case of Feature Type Space, this means making instances of spaces such as hall, living, kitchen, veranda, et cetera. The Feature Instance Living is defined as follows:

```plaintext
Elements.spatial.Space Living = {  
  Descr ("Living room");  
  contains[1] = Dining;  
  contains[2] = Sitting;  
  function = FunctionLiving;  
  height = LivingHeight;  
  daylightUsed = DaylightLiving;
}

Elements.spatial.Space Dining = {  
  function = FunctionDining;  
  height = LivingHeight;  
  numberOfPersons = NumberOfPersonsInDining;
}

Elements.spatial.Space Sitting = {  
  function = FunctionSitting;  
  height = LivingHeight;
}

Elements.spatial.Function FunctionLiving = {  
  Value ("Living")
}

Elements.spatial.Function FunctionDining = {  
  Value ("Dining")
}

Elements.spatial.Function FunctionSitting = {  
  Value ("Sitting")
}

Geometry.dim.Height LivingHeight = {  
  Value (2.7)
}

General.value.Number NumberOfPersonsInDining = {  
  Value (8)
}

Arch.value.Daylighting DaylightLiving = {  
  Value (True)
}
```
In subsequent phases the Feature model is established on the basis of the previous phase, which enables to track changes during the design process.

8.9.2 Feature-based description of phase 0

The brief is translated into a Feature model. It consists of elements of the design (spaces and objects, like bathtub, toilets, etc.) and of relations between elements. Many relationships are defined as constraints, in particular those that concern spatial relationships. Topological relationships can be expressed as: A adjacent to B, A in B, A above B, and their logical opposites A not adjacent to B, A not in B, A not above B, with A and B Features of the type Space.

Other constraints concern access from one space to another. These access constraints can be expressed as: A accesses B and the opposite A does not access B. Since this relation is reciprocal, only one access constraint has to be defined.

The third type of constraint is expressed in the statement “kitchen is closed with respect to the living,” meaning that there is no visual relation between the kitchen and living space. The constraints A visual to B and the opposite A not visual to B are reciprocal, and can be defined in the same manner as the access relation.

The ‘fireplace in living’ constraint can be expressed in two ways: as a Fireplace_in_Living instance of a Heating_in_Space constraint Feature Type, or by establishing an association-relation for the Feature Instance Living with the Feature Instance Fireplace. Since heating elements are bound to occur in more spaces, the first option is chosen.

The tables below show the Feature Types and Feature Instances for elements and constraints in phase 0 of the case study respectively.

Table 8.1 Feature Types and Feature Instances of elements in phase 0.

<table>
<thead>
<tr>
<th>Feature Types (supertypes in brackets)</th>
<th>Feature Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space</td>
<td>Hall, Toilet, Wardrobe, Living, Sitting, Dining, Kitchen, Veranda, Scullery, Garage, Bedroom, Bathroom, Shower, GuestRoom</td>
</tr>
</tbody>
</table>

Modelling Architectural Design Information by Features
The application of Features in an architectural design case

### Table 8.2 Feature Types and Feature Instances of constraints in phase 0.

<table>
<thead>
<tr>
<th>Feature Type (constraint type in brackets)</th>
<th>Feature instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space_adj_to_Space (spatial)</td>
<td>Kitchen_adj_to_Living,</td>
</tr>
<tr>
<td></td>
<td>Veranda_adj_to_Living,</td>
</tr>
<tr>
<td></td>
<td>Scullery_adj_to_Garage,</td>
</tr>
<tr>
<td></td>
<td>Scullery_adj_to_Bedroom,</td>
</tr>
<tr>
<td></td>
<td>Scullery_adj_to_Kitchen</td>
</tr>
<tr>
<td>Space_adj_to_Garden (spatial)</td>
<td>Kitchen_adj_to_Garden,</td>
</tr>
<tr>
<td></td>
<td>Bedroom_adj_to_Garden</td>
</tr>
<tr>
<td>Storey_above_Storey (spatial)</td>
<td>Storey1_above_Storey0</td>
</tr>
<tr>
<td>Furniture_not_in_Space (spatial)</td>
<td>Furniture_not_in_Kitchen</td>
</tr>
<tr>
<td>Heating_in_Space (spatial)</td>
<td>Fireplace_in_Living</td>
</tr>
<tr>
<td>Stair_not_in_Space (spatial)</td>
<td>Stair_not_in_Living</td>
</tr>
<tr>
<td>Space_Space (access)</td>
<td>Bedroom_accesses_Bathroom</td>
</tr>
<tr>
<td>Space_Garden (access)</td>
<td>Bedroom_accesses_Garden,</td>
</tr>
<tr>
<td>Space_not_visual_to_Space (visual)</td>
<td>Kitchen_accesses_Garden</td>
</tr>
<tr>
<td></td>
<td>Kitchen_not_visual_to_Living</td>
</tr>
</tbody>
</table>
Note that the above collection of Feature instances from the model for phase 0 of the design does not represent a complete specification. Actually, the majority of specifications of a design are normally not included in the brief, but implicitly assumed. The actual brief only indicates the special requirements gathered from the communication with the client, like the access from the bedroom to the garden, while leaving out the more trivial requirements, such as access from kitchen to living. Also, the minimum areas of spaces were not included in the brief. In some cases, this kind of requirements was expressed by the number of persons expected to use a space.

In part of the Feature model of phase 0, relationships are modelled that are not defined by the Feature Types. The reason for this is that the relationships in question are not considered typical for the particular Features, at least not at this point in the design process. It is therefore decided to model these relationships as instance level relationships only.

An example of such an incidental relationship is shown in figure 8-16.

Figure 8-16 Example of an incidental relationship that is modelled at the instance level only. The roof component of the Veranda is not defined for the Space Feature Type. The Veranda_adj_Living Feature is a constraint between the veranda and the living.

The figure shows typologically defined relationships, such as those defined by the adjacency constraint (here modelled as a Complex Feature Type), but also demonstrates that the veranda, an instance of the Space type, relates to a Roof instance. The Space type does not define such a relationship, therefore it must be added at instance level. The description of the Veranda Feature and its related Features is given below.

```plaintext
Elements.spatial.Space Veranda = {
    Descr {"Veranda"},
    function = FunctionVeranda,
    daylightUsed = DaylightVeranda;
    roof = RoofVeranda;
}
Elements.spatial.Function FunctionVeranda = {
    Value {"OutdoorDining"}
}
```

Modelling Architectural Design Information by Features
The application of Features in an architectural design case

The Roof type and its instance for the veranda are defined as specified below:

```java
class Elements.spatial.Roof {
    Spec Elements.material.Material material;
    Spec Arch.value.Daylighting daylightUsed;
}

class Elements.spatial.RoofRoofVeranda {
    material = MaterialRoofVeranda;
    daylightUsed = DaylightVeranda;
}
```

The adjacency constraint between the veranda and living room is modelled by the prototype as a complex Feature; its type description (`Space_adj_Space`) is given in the previous subsection. The Feature Instance looks like this:

```java
class Constraints.spatial.Space_adj_Space {
    Veranda adj_Living = {
        access[1] = Veranda;
        access[2] = Living;
    }
}
```

However, should the prototype have provided full support of the FBM framework, including the Constraint Feature Types, this adjacency would have been defined as shown below and in figure 8-17.

```java
constraint Constraints.spatial.Space_adj_Space
    (Elements.spatial.Space space1, Elements.spatial.Space space2) {
        TypeConstraint {adjacency}
    }
```

![Diagram](image)

**Figure 8-17** The adjacency constraint type and instance as modelled according to the FBM framework.

The adjacency of the veranda and living room would have looked like is shown in figure 8-17 with the following notation:

```java
class Constraints.spatial.Space_adj_Space {
    Veranda adj_Living = {
        (Veranda, Living) = {
        }
    }
```
8.9.3 Feature-based description of phase 1

In phase 1, the brief is transformed to a set of spaces in a drawing. Significant changes with respect to phase 0 consist of assigning shapes and their dimensions to spaces, and locating them in the site by means of a grid.

As stated before, in the case analysis there are no Geometric Feature Types, because of the limitations of the Feature Tool prototype. Therefore, shapes are defined as Complex Feature Types. The notion of shape is very basic in architectural design, and many other kinds of shapes may occur later in the design, each with their own intrinsic properties. A supertype 2DShape is defined of which Rectangle is a subtype. For the definition of a rectangle and its position in a drawing, the dimensions and position need to be defined as Feature Types as well. Therefore, Length, Point, and Coordinate also are new Feature Types. Since the initial plan is orthogonal, the orientation of shapes is not considered in phase 1.

```plaintext
complex Geometry.shape.2DShape {
    TypeDescr (*General 2D shape with reference point*)
    Spec Geometry.top.Point referencePoint[1..1];
}

complex Geometry.shape.Rectangle(Geometry.shape.2DShape) {
    TypeDescr (*Rectangular shape with dimensions and reference point*)
    Spec Geometry.dim.Length length[1..1];
    Spec Geometry.dim.Length width[1..1];
}

real Geometry.dim.Length {
    TypeDescr (*Linear dimension in m*)
    TypeDefault {1}
    TypeUnit {"m"}
}

complex Geometry.top.Point {
    TypeDescr (*Point in 3D orthogonal axial system*)
    Spec Geometry.top.Coordinate x[1..1];
    Spec Geometry.top.Coordinate y[1..1];
    Spec Geometry.top.Coordinate z[1..1];
}
```

Figure 8-18 Definition of the 'geometric' Feature Types, defined as Complex Feature Types.
The Feature Type 2DShape is associated with the existing Feature Type Space. The line in bold-face shows the addition to the Feature Type from phase 0:

complex Elements.spatial.Space (
    TypeDescr {"Space element where activities can take place"}
    Has Elements.spatial.Space contains[0..?];
    Spec Arch.value.Daylighting daylightUsed;
    Assoc Elements.spatial.Function function;
    Spec General.value.Number numberOfPersons;
    Spec Geometry.dim.Height height;
    Assoc Geometry.shape.2DShape shape;
)

Figure 8-19 Redefinition of the Feature Type Space in phase 1, with addition of the shape component.

The shapes are drawn in a grid which is used for co-ordination. The grid can be defined by stating its module and a point of origin relative to which co-ordinates are established (this also accommodates the use of multiple grids later in the design case). Both the origin of the grid and the position of elements in the grid require a set of co-ordinates. Therefore, at the instance level, a Feature Instance Origin (instance of Geometry.topology.Point) is defined relative to which measures can be taken and grids positioned. The left-bottom corner of the grid instance is placed on the origin.

Figure 8-20 Definition of the Feature Type Grid.
The positive x-axis is oriented horizontally and to the right of the origin. The positive y-axis is oriented vertically and above the origin, as is customary in architectural design. For the Feature Type Rectangle, the reference point is defined as the left-bottom corner of the rectangle, width and length being measured in the orientation of the positive x and y axis.

These Feature Types and Feature Instances suffice to describe the state of phase 1. The Feature Instance Kitchen, for example, is changed because of the additional association to the Feature Type Space of which it is an instance, and the definition of its location and dimensions in the associated Rectangle. In phase 1, the Kitchen has dimensions 3.60 m x 3.60 m, and is located at the co-ordinates (6.0, 6.0, 0). Again, the line in bold-face shows the addition to the Feature Instance from phase 0:

```plaintext
Elements.spatial.Space Kitchen = {
  Descr {"Kitchen"},
  daylightUsed = DaylightKitchen;
  function = FunctionKitchen;
  shape = Rectangle_Kitchen;
}
Geometry.shape.Rectangle Rectangle_Kitchen = {
  Descr {"Rectangular shape for kitchen"},
  referencePoint = RefPoint_Kitchen;
  length = Length_Kitchen;
  width = Width_Kitchen;
}
```

Note that the component referencePoint of the Rectangle type is inherited from its supertype 2DShape.
In phase 1, not all spaces mentioned in phase 0 are present, and there are four spaces that have not been assigned a function by the architect. The older instances of spaces that are included in phase 1 change in the same manner as the kitchen example. Newly introduced spaces are instantiated directly according to the new Feature Type definition.

These two steps in the case study demonstrate that an actual design case can be described by Features, and that the Feature framework can accommodate changes both on the type level and the instance level.

8.9.4 A classification of design actions

The description of the design process in the case studies provides a new way to look at design processes [Achten and van Leeuwen 1999]. In particular, changes from one phase to the next can be expressed in terms of changes in the Feature model. In this way, design actions can be matched to Feature model alterations. These, in turn, can then be used for the specification of the development of dedicated design support tools.

The Feature model alterations that occurred in the case study are very specific for the design case that is studied. Therefore, it is necessary to classify them into more general descriptions of design actions and associated changes in the Feature model. The classification provides the proper vocabulary for establishing the functional brief for the VR-DIS system and provides a concrete filling-in of the discussion in section 8.4 on manipulation of Feature Types and Feature models. The following table presents the classification and the definition of the terms for changes in the Feature model that resulted from the analyses of the case study.
Table 8.3 Design actions and changes in the Feature model.

<table>
<thead>
<tr>
<th>Design action</th>
<th>Changes in the Feature model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generalisation</td>
<td>When a group of objects share common properties, define the specific objects as Feature Types, and define a Feature Type (supertype) of which they are subtypes. The supertype functions as generalisation.</td>
</tr>
<tr>
<td>Concept identification</td>
<td>Terms in the brief that are relations or spatial-, material-, and functional elements, are defined as Feature Types.</td>
</tr>
<tr>
<td>Element creation</td>
<td>Terms in the brief that are actual parts in the design (hall, floor, fireplace) can be instantiated directly on the basis of the corresponding Feature Types.</td>
</tr>
<tr>
<td>Constraint creation</td>
<td>Terms in the brief that are relations in the design can be instantiated on the basis of Constraint Feature Types.</td>
</tr>
<tr>
<td>Concept extension</td>
<td>Adding an association relation to a Feature Type in order to include more characteristics.</td>
</tr>
<tr>
<td>Shaping</td>
<td>Giving shape to the spatial elements involves element creation of the Feature Types Shape and of Feature Types Position and Dimension.</td>
</tr>
<tr>
<td>Assignment</td>
<td>At the Instance level make an association relation between Feature Instances.</td>
</tr>
<tr>
<td>Move</td>
<td>Move means that the co-ordinates that define a position have been changed in a Feature Instance.</td>
</tr>
<tr>
<td>Substitution</td>
<td>Substitution means that an existing association between Feature Instances is broken and that one of the Feature Instances is replaced.</td>
</tr>
</tbody>
</table>

8.9.5 Conclusions from the case study

In [Achten and van Leeuwen 1998], the following conclusions were drawn from this case study:

1. Using the Features approach to describe the design brief results in a description of this first stage in the design process that is much more precise and comprehensive than traditional textual descriptions.

2. The flexibility of the Features approach allows capturing the changes of the design during the various stages of the design process.

3. The same flexibility, however, also allows a great number of ways to establish the design model, of which the most optimal one becomes apparent only towards the end of the process.

4. Availability of the Generic Feature Types and the compilation of collections of Specific Feature Types representing the particular designer's style are therefore prerequisite for efficient application of the Feature modelling approach.

5. The interface to the Feature data must effectively facilitate the control of both Feature Type Libraries and Feature models, as their complexity will grow considerably during the course of design.
Integrity management of both Feature Type Libraries and Feature models is an important issue for the design system which has not been addressed sufficiently in this stage of prototyping.

In the case, nine design actions are identified and described in terms of changes in the Feature model. They are either at Feature Type level only (generalisation, concept identification, and concept extension), at the Feature Instance level only (element creation, constraint creation, assignment, move, and substitution), or a mix of both levels (shaping). The classification of design concepts deduced from these design actions can be used to define the design functionality of the VR-DIS system in terms of changes in the Feature model [Achten and van Leeuwen 1999].
Modelling Architectural Design Information by Features
Conclusions and Discussion

Conclusions

This research project started with a study on what comprises architectural information and why it should be modelled. It continued with an investigation of various approaches of modelling architectural information, in particular the approach of product modelling, and identified the specific problems that arise when modelling architectural design information.

In chapter two, it is observed that the general approach developed in the area of product modelling in the Building & Construction industry does not suffice for the purpose of modelling architectural design information. The main cause for this is that product models are generally not developed for the support of architectural design. They comprise a rigid, pre-defined structure, often with complex hierarchies that do not allow addition and modification of object definitions, nor ad hoc, unforeseen correlations between objects. Chapter 2 ends with the conclusion that, for an adequate support of architectural design, it is necessary that a designer can dispose of an accurate representation of the state of design during the design process, and that this representation is able to contain the semantics of the design paradigm and the meaning of design decisions.
Conclusions and Discussion

In chapter three it is acknowledged that design has a dynamic nature in the way it deals with information. Mainly due to the problem-solving character of design, but also because of the development of designers during a project and during their career, designers are not helped with static tools, but require tools that can be adapted to situations and insights that change during design tasks. The conclusion from this observation is that the content and structure of information required and produced during design cannot be presupposed. Design requires an evolutionary approach to modelling information. This requirement is translated into two essential characteristics of design information models: extensibility and flexibility. Extensibility relates to the ability of the conceptual model to be extended with new definitions of information entities to represent emerging concepts. Flexibility, other than that required for extensibility, relates to (a) the ability of conceptual models to evolve along with the evolution of a designer's insights, and (b) the ability of actual models of design to represent concepts that are somehow divergent from available conceptual definitions. Examples of the latter are the omission of part of the concept's definition or addition of attributes that are especially relevant in the current context of the particular concept. The methodology for representation of information in a design support system must provide for this kind of typological extensions and ad hoc information modelling.

The first research question stated in this thesis was:

What should be the performance of (a) a model for architectural design information, (b) its structure, and (c) the modelling environment to support this structure?

According to the above conclusions, the answers to the three parts of this question must read:

(a) An architectural design information model should perform in such a manner that it reflects accurately the design rationale intended by the designer, at all stages of the design process: the model should evolve along with the proceeding state of the design.

(b) In order to achieve this, its structure should allow for extensibility in terms of definable information entities and flexibility in the way entities can be interrelated in the desired organisation.

(c) The modelling environment to support this kind of structure must (1) provide the proper interfaces for defining information content and structure; (2) handle both abstract types of information and information concerning tangible concepts; (3) provide design support tools, such as reasoning and inference mechanisms, that take in account the extensions of the underlying conceptual model and the flexibility manifested in the model's structure.

The review in chapter four shows that Feature-based modelling (FBM) approaches, developed for the area of mechanical engineering, concentrate mainly on form Features. Research on FBM is very much geometry oriented, aiming to integrate part design and
part manufacturing processes. It also focuses on the lowest level of product definition, namely that of the individual parts of the product. Nonetheless, some of its key concepts are very relevant for the architectural design context:

1. Features are used to model the rationale of design, because they represent the semantics of design in the terminology of the designer.
2. Being the formalisation of design terminology, Features act as the primitives for defining a product and for reasoning about a product and thus bring the modelling process closer to supporting the design process.
3. The structure of a Feature model is not rigidly predefined, but is established during the process of design. This is possible, because Features are defined as relatively independent entities that have an autonomous behaviour and function in the model.
4. The Feature modelling paradigm supports extension of conceptual models by definition of new classes of Features. The tools for design can be adapted in this manner to the specific needs of each individual design case, enlarging the library of available resources for design.

These issues are of interest for architectural design, not only at the most detailed level of design, but for the architectural product, the building, as a whole. This has led to the following, additional key-concepts of Feature-based modelling in an architectural design context:

5. Features are used for modelling at multiple levels of abstraction in architectural design.
6. Features are used for modelling both physical and non-physical concepts in architectural design.

Together with the inventory of architectural design information in chapter 6, the above key-concepts have formed the basis for the development of the FBM framework in chapter 7.

The infrastructure of the framework has a three-layered architecture, of which the top layer, the Meta Layer, defines the format of the other two layers for Feature Types and Feature Instances respectively. The Meta Layer, of which the details are defined in chapter 7, facilitates the definition of new Feature Types and their corresponding Feature Instances in the framework. This is possible because both the Feature Types and Feature Instances are objects of the classes defined in the Meta Layer, which provides the framework with a generalised method to access both Feature Type data and Feature Instance data. The framework thus supports the requirement of extensibility of conceptual information models for architectural design.

The desired flexibility of Feature models requires that the structure of the model is not rigidly predefined but can, in part, be established during the modelling process, concurrently with the design process. This flexibility is achieved by the ability of the

*Modelling Architectural Design Information by Features*
framework to allow the definition of relationships between Feature Instances that are not defined at the level of Feature Types. Instance level relationships eliminate the need for unintended actions at the level of Feature Types, such as modification of existing types and creation of subtypes or copies of types. Actually, the capability of instance level relationships allows a more accurate representation of the state of a particular design, for instance in those cases where the application of a particular design concept does not completely conform its initial definition. It does not urge the designer to revise the concept definition, nor to conceive a new concept for each divergent situation.

The only entities that occur in Feature Type Libraries and Feature models are Feature Types and Feature Instances. Attributes of these entities are defined as relationships rather than embedded definitions. The attributes of a Feature Type are defined as relationships to other Feature Types or Feature Instances, the attributes of a Feature Instance are relationships to other Feature Instances. As a result, Feature models and Feature Type Libraries can be seen as graphs, with the Feature Instances and Feature Types as the nodes of the graph and the attributes and relationships between them as the arcs of the graph. This graph structure of the Feature models and libraries has several advantages.

1. Using references rather than multiple occurrences of attribute definitions in Feature Types reduces redundancy to a minimum. From the designer's point of view, this 'sharing of attribute definitions' makes modification of Feature Types more simple and potentially enhances consistency, however special functionality of the modelling system is required to support this. If the definition of the same attribute in different contexts is actually defined as multiple references to a single Feature Type, this represents more accurately the intention of the definition.

2. Similarly, using multiple references for the attributes of Feature Instances where the same attribute is actually intended represents more accurately the design rationale. For example, the same height of all the doors in a building can be modelled as a 'height' Feature that forms a shared relation with all the 'door' Features. This 'sharing' paradigm eliminates the need for adding constraints to the model to ensure that all doors will maintain the same height. Doors with a different height can still refer to other height Features.

3. Relationships between Feature Instances that are added to the model at the instance level are also references and thus similar to the attributes defined at the type level. This allows both the system and designer to treat them in the same manner.

The fulfillment by the proposed FBM framework of the requirements of extensibility and flexibility as described above facilitates its support of the dynamic nature of design.

The API developments in the three subsequent prototyping efforts have shown that the framework can be successfully implemented. The second prototype has been developed into a working tool for Feature Type definition and Feature modelling, and has proved to work satisfactorily in the design case study, as is discussed below. This
Conclusions and Discussion

Prototype confirms the adequacy of the framework's functionality with respect to the requirements of extensibility and flexibility.

The third prototype has concentrated on the implementation of the API with respect to the functional requirements defined in the context of the VR-DIS project. Currently it forms the basis for the development of a new version of the Feature Tool, similar to that of prototype 2, and a much more enhanced design tool embedded in a Virtual Reality design environment. This prototype provides the evidence that the functionality of the classes for Feature Types and Feature Instances, defined in the Meta Layer of the framework, can be implemented into a sound basis for the development of an innovative design system for support of dynamic design tasks.

The Feature Types that have resulted from the case study on architectural design by Features can be assessed as relatively trivial. Several reasons for this can be given. This case study was the first actual application of the Feature modelling approach in an architectural design context. Therefore, the chosen design case is a relatively simple case and is not worked out in great detail\(^1\). Since no library was available with Generic Feature Types, even the most elementary Feature Types had to be defined from scratch. However, the case study has demonstrated that very significant problems in design, such as the transition from functional design brief to spatial layout, can be addressed with the Feature modelling approach. Even with the limited functionality of the phase 2 prototype, this case study has validated, in an actual design context, the principles of the FBM framework and provided evidence for the adequacy and effectiveness of the framework's functional design. This confirms the effective application in the architectural context of the key concepts 1 to 4 of the FBM approach that are mentioned on page 231 and provides sufficient indications of the validity of the additional key concepts 5 and 6.

In conclusion, through the usage of the Feature Tool, the case study has modelled the semantics of the design case in the terminology that was formalised in parallel with the modelling process. The model evolved, as the different design stages were analysed. It addressed multiple levels of abstraction and both physical and non-physical concepts of the building design.

The second research question can thus be answered:

*Considering the required performance, how should an information model for the support of architectural design be structured?*

The prototyping for the implementation of the API and the experiments with the results thereof in the design case study have provided the indications that the

---

\(^1\) However, it should be noted that, of the thirty phases that were included in the design case study, only a small part of the first two phases is registered in section 8.9.
Conclusions and Discussion

information modelling approach as developed in the FBM framework meets the considerations on required performance that followed from the first research question. The definitions of the classes of Feature Types and Feature Instances and the possibilities offered for structuring the libraries and models, have proved to be a satisfactory method of modelling information for the support of architectural design.

Discussion and future work

Generic Feature Types and standardisation

One of the conclusions from the design case study in section 8.9 confirmed the necessity of predefined Generic Feature Types. The role of these Generic Feature Types in relation with standardisation efforts has been discussed in section 8.5. The inventory of Feature types in chapter six has resulted in an understanding of what kind of information is relevant when modelling an architectural design. This understanding was necessary to develop the information modelling approach in the FBM framework. Yet, the categorisation of Features as shown in figure 6-6 on page 114 also forms a proposal for the taxonomy of standardised Generic Feature Types. The same taxonomy can be used for the organisation of Specific Feature Types defined by the individual designer. Further investigation is necessary both on the taxonomy of Feature Types and on the detailed definition of Generic Feature Types for the various disciplines in the Building & Construction industry. The Generic Feature Types will play an important role in the integration of knowledge and evaluation tools in design systems.

One of the key concepts in the FBM framework is the ability to model relationships between Feature Instances that have not been foreseen in the corresponding Feature Type definition. For automatic interpretation of these relationships, a design system can only make use of the type of relationship that was chosen; knowledge concerning the detailed semantics of the relationship cannot be accessed without interpretation of the role name of the relationship. In section 8.2, it is concluded that further prototyping in the development of design systems will have to demonstrate whether the restricted knowledge about the type of relationships alone is sufficient for the level of reasoning that is required for design systems, or that knowledge and usage of standardised role names is essential. Another option is to extend the number of four role types that are currently defined in the framework, in order to allow more detailed distinctions between the relationships without the need for interpretation of role names.

Feature Type definition and Feature instantiation

In section 8.1, various procedures are mentioned for the definition of Feature Types. The procedure of defining a new type from scratch has been documented in detail. This procedure in fact formalises the conceptual clustering strategy for classification (see page 182). The other two procedures that are mentioned are based on the prototype-approach to classification and on the classical categorisation approach. The implications of these
Conclusions and Discussion

procedures have not yet been investigated, but their importance in a design system is beyond question.

Similarly, the different scenarios for instantiation of Features, discussed in section 8.3, require further investigation of their implications for the functional design of modelling systems. One of these approaches relates to the Feature recognition methodologies developed in mechanical engineering. This approach involves the matching of structures of Features that were modelled using instance level relationships with the definitions of Feature Types. Feature structures at instance level that are sufficiently similar to known Feature Types can be replaced with instances of these types, which may add significant knowledge and semantics to the design model. A crucial factor for a successful development of this methodology is the formalisation of heuristics for the matching of Feature structures and known Feature Types. Another aspect is the development of heuristics for alteration of Feature structures to match existing Feature Types. Research on this topic has been initiated in the VR-DIS research programme.

Future work within the VR-DIS research programme

The developments in the VR-DIS project are still in an elementary stage. Many foreseeable aspects must be addressed, many of which have to do with the management of the underlying Feature data. The development of the framework into the various prototypes has hitherto excluded the implementation of geometric, constraint, and handler Features. For the initial evaluation and assessment of the framework this is not considered an unacceptable shortcoming, however, these Feature Types are crucial for the full development and utilisation of a Feature-based design system, for example for the implementation of derived attributes of Features, integrity management rules, and automated manipulation of Features on the basis of occurring events (see pages 150 and 152).

In the first VR-DIS prototype, spatial constraints have already been incorporated in the system, albeit in a limited manner by defining them as Complex Feature Types. An integrated constraint solver module was programmed to interpret predefined Feature Types representing the constraint data. This functionality must, in future prototypes, be uncoupled from the contents of the Feature Types, but rather be based on the structure of Constraint Feature Types. A generic approach for doing this is by standardising the protocol for communication between constraint solving/checking modules and the Feature manager module (see also figure 8-9 on page 208). This standardised protocol must provide generic mechanisms for triggering constraint solving and checking and for communicating the outcome of the solving process back to the Feature model. The structure of Constraint Feature Types, as defined in the framework, forms the basis for such a standardised protocol.

While form Features have full attention in mechanical engineering, in this research project they have been almost disregarded. Although this can be justified because of the focus in this research project on the entirety of architectural design information and the general approach to structuring the modelling thereof, the aspect of shape is evidently a
 Crucial factor in the development of any design system. Similar to the provisional way of implementing constraints through Complex Feature Types, the VR-DIS prototype has also included geometric data in Complex Feature Types. Again, the interaction between the geometric modelling engine and the Feature data relied on the structured contents of the 'geometric' Complex Feature Types, which, in future prototypes, must be uncoupled and based on the structure of Geometric Feature Types. In the envisaged VR-DIS system, a single geometry engine and a single user interface are devised, which relaxes the design of the communication protocols between these two modules and the Feature management module (see figure 8-9 on page 208).

Event handling has not yet been incorporated in the prototyping efforts at all. The issues for further research on this topic not only include the design of the procedural language for event handlers. Event handlers will play an important role in the functionality of the design supportiveness of the system, especially in tasks such as instantiation and manipulation of collections of Features. Establishing relationships between newly instantiated Features and the existing Feature model can become a very tedious task, much of which can be taken over by event handlers. A conceivable example of such a scenario is the instantiation of a 'space' Feature that automatically leads to the instantiation of related 'function' and 'wall' Features, while the 'height' component of both space and walls is automatically related to the 'height' component of the active 'building-level' Feature. Further development of the event handling paradigm in the framework and prototypes requires detailed specification of the requirements for the procedural language for event handling. Based on these requirements, existing technology, such as scripting languages, must be assessed.

The above described issues of research will be conducted within the VR-DIS research programme, which comprises many other issues that are addressed by the various members of the VR-DIS research team. This includes interface development for architectural design in VR, abstract data-mining in VR, incorporation and evaluation of user-behaviour, communication in interdisciplinary design using agent technology, and integration of design knowledge from various architectural disciplines. The integrated approach to these and the above described research issues in the VR-DIS programme, aiming at the development of architectural design systems, will deliver the final evidence of the value of the Feature-based modelling approach for the support of architectural design tasks.
Appendix A
EXPRESS-G: graphical notation technique

The Feature-based information modelling framework presented in this thesis defines a three layered information infrastructure. The top layer of this infrastructure is the so-called Meta Layer, which defines the format of the two lower layers. The entities that compose this Meta Layer are graphically presented in chapter 7 using the graphical notation technique EXPRESS-G. This appendix explains this technique.
EXPRESS-G is the graphical counterpart of the textual data specification language EXPRESS which is defined in the international standard ISO 10303 part 11 [ISO TC184 1994c]. This document is part of the standardisation work by ISO on 'Industrial automation systems – Product data representation and exchange', which is better known by the name STEP (STandard for the Exchange of Product data).

This appendix summarises the various elements of the EXPRESS-G notation that are used in this thesis. It should be noted that this is a subset of the complete set of symbols defined by EXPRESS-G. In order to enhance the readability of the diagrams using this notation technique, some additions have been made by the author. These additions are introduced and indicated in the text below where appropriate.

What is referred to in this thesis as a conceptual model, is called a schema in EXPRESS. Diagrams in EXPRESS-G show the graphical representation of definitions either at the level of schemata, showing schema-relationships, or at the level of entities of a schema. Only entity-level diagrams are included in this thesis. A schema may be too large for representation in a single diagram, therefore multiple diagrams can be used to represent a single schema.

**Definition of symbols**

EXPRESS-G shows the definition of 'things' by means of rectangular boxes and relationships among things by lines joining the boxes. Form and line style of the boxes and lines provide extra information on the kind of definition or relationship. The name of a thing is noted inside its box, whereas the role name of a relationship is written close to the relationship line.

**Data Types**

EXPRESS defines four groups of data types:

- **Simple data types**
  This group includes such data types as String, Integer, and Real.

- **Aggregation data types**
  Aggregation data types are used to define collections of data types. This group includes types such as Array, Set, and List. Graphically, the Aggregation data types are noted by means of relationship lines.

- **Named data types**
  A Named data type is either an Entity data type or a Defined data type. A Defined data type which simply assigns another name to an existing Defined data type or to a Simple data type. An Entity data type defines the properties of a class of objects.

- **Constructed data types**
  These are either Enumeration or Select data types.
Symbols for Simple data types

The simple data types that are used from those provided in the EXPRESS language, are: String, Number, Integer, Real, and Boolean. The String data type, in this case, has as its domain character-sequences of varying width. The Integer data type has the domain of all integral numbers, whereas the Real data type includes rational, irrational and scientific real numbers. The Number data type is used for those cases where a more specific indication of the data type, integer or real, cannot be given. Finally, the Boolean data type is used for representation of either the literal TRUE or FALSE. The simple data types are graphically noted as shown in the figure below.

![Figure A-1 Symbols for EXPRESS simple data types.](image)

Symbol for Defined data types

Defined data types provide names for other, existing data types which are either Simple data types or other Defined data types. They are graphically represented by a box with a dashed line style as is shown in the following example.

![Figure A-2 Example of using the symbol for the Defined data type, together with the Simple data type that it provides a new name for.](image)

Symbols for Constructed data types

Constructed data types are the Enumeration data type and the Select data type. Since these data types can only be used in EXPRESS as the underlying representation of Defined data types, only the graphical notation of the combination of a Defined data type with an Enumeration or Select data type are used in this research project. These are called Defined Enumeration data type and Defined Select data type respectively.

An Enumeration data type has as its domain an ordered set of identifying names. This domain is normally not graphically shown in EXPRESS-G. Yet, for reasons of clarity, it is included in the diagrams in this thesis and indicated with an arrow pointing to the Enumeration symbol.
Appendix A

A Select data type has a domain that is the union of the domains of the named data types in its select list. Graphically, a Select data type is shown along with its select list of named data types. The symbols of the select list are connected to the symbol of the Select data type by a line with branches to each of the symbols.

Figure A-3 The symbol for the Defined Enumeration data type with the added domain of identifiers (left) and the symbol for the Defined Select data types shown with the data types composing its select list (right).

Symbol for Entity data types

An Entity data type, or entity in short, represents a class of objects which have common properties. The properties of an entity are represented as attributes in EXPRESS. In EXPRESS-G, entities are represented by showing the decomposition into their attributes as relationships to other symbols. The entity itself is represented by a box enclosing the identifier of the entity.

Relationship symbols

As mentioned above, the attributes of an entity are defined by the relationships of this entity with other data types. Generally, two kinds of relationships are distinguished: inheritance relationships and all other relationships. The inheritance relationships, defining that an entity is a subtype of another entity, are graphically indicated with a thick line marked with an open circle at the end of the subtype. Inheritance relationships do not have a role name.

Figure A-4 Example of an inheritance relationship. Both Male and Female are subtypes of the Entity data type Person. The '1' indicates that any instance of Person can be only one of the subtypes Male and Female, not both at the same time.

Other relationships are indicated with a thin line. Relationships are in principle bidirectional, yet one direction is emphasised in EXPRESS. This emphasised direction is marked with an open circle. The role name that may be written next to the relationships
line also indicates this emphasised direction of the relationship (see the example in figure A-5). Optional relationships are shown with a dashed line.

Aggregation data types are defined by means of attributes of entities. In the graphical notation, this is indicated by adding a letter to the role name of a relationships (e.g., L for List, S for Set), and, in square brackets, the cardinality of the aggregation. A question-mark indicates an unlimited upper bound.

![Diagram](image)

**Figure A-5** Example of the notation of relationships. A Person must have a name, which is an attribute of the type Name, a Defined data type with underlying type String. The yearOfBirth is an optional attribute, and a Person has 1 or more addresses, which are stored in a list of Strings.

**Composition symbols**

Because a data definition schema may be too large to be represented in a single diagram, the schema can be split up into multiple diagrams. For this purpose symbols are required to allow references between the diagrams of a schema. These symbols are called Page references and consist of a rounded box. Page references occur both in the referring diagram and the referred diagram. The referring symbol contains the number of the diagram, the number of the reference in that diagram, and the identifier of the referred data type. The referred symbol contains the number of the diagram, the number of the reference, and, in parentheses, a list with the numbers of all the diagrams that refer to that symbol. The role name of the reference is shown in the referring diagram. The open circle marking the direction of the relationship is placed in the referred diagram, at the referred symbol.

The EXPRESS-G definition allows only referencing for non-inheritance

![Diagram](image)

**Figure A-6** Example of page-references from one diagram to another. Diagram 1 on the left refers to the Name and Address, respectively numbered 1 and 2, in diagram 2.
relationships. However, the diagrams in this thesis do contain references in inheritance relationships. In these cases, the label [specialisation] is added to the rounded box of the referring symbol, where normally the identifier of the referred symbol would appear.

Another kind of Composition symbol is needed when data types defined in one schema are referred to in another schema. These symbols are called Inter-schema references and are referring symbols only. They consist of a rounded box inside a rectangular box. EXPRESS allows two kinds of inter-schema references, namely for referencing data types and for using data types. The latter is not used in the diagrams in this thesis. A reference to a data type in other schema is indicated by a rounded box inside a rectangular box, where the rectangular box has a dashed line style. In the rounded box, the name of the referred schema is written, followed by the name of the referred data type. Under the rounded box, an alias for that data type in the referring schema may be included.

![Diagram](image)

Figure A-7 Example of an inter-schema reference. A Person is located in an Office, which is an alias for a data type called Space that is defined in the schema Building.
The syntax of the textual notation of Feature Types and Feature Instances is given in this thesis in a derivative of the Wirth Syntax Notation (WSN) [ACM 1977]. This appendix specifies the syntax of the WSN notation.
The syntax of the Wirth Syntax Notation technique is defined, in itself, in the table below:

<table>
<thead>
<tr>
<th>Syntax for specification of the textual notation technique, noted in itself.</th>
</tr>
</thead>
<tbody>
<tr>
<td>syntax = ( production ) .</td>
</tr>
<tr>
<td>production = identifier ' = ' expression ' ' .</td>
</tr>
<tr>
<td>expression = term { '</td>
</tr>
<tr>
<td>term = factor ( factor ) .</td>
</tr>
<tr>
<td>factor = identifier</td>
</tr>
<tr>
<td>literal = ''' character { character } ''' .</td>
</tr>
<tr>
<td>group = '(' expression ')' .</td>
</tr>
<tr>
<td>option = '[' expression ']' .</td>
</tr>
<tr>
<td>repetition = '{' expression '}' .</td>
</tr>
</tbody>
</table>

Explanation of the elements of the notation technique

A production is the definition of an element on the left of an equal sign ' = ', which is
defined to be the combination of the elements on the right of the equal sign. The spaces
in the production bear no meaning, unless they are part of a literal. A production may
continue over multiple lines and a period '.' denotes the end of a production.

Identifiers on the right of a production, e.g. 'factor' in the 4th line of the above
syntax notation, are non-terminal symbols that appear on the left in other productions.
They may consist of letters and underscores '_'. Identifiers are not case sensitive.

A literal is a terminal symbol, not further defined, which is a case insensitive
sequence of characters enclosed in single begin and end quotes. A single quote can be
included in a literal by preceding it with a backward slash '\'. A character, in this context,
is any character as defined by ISO 10646 cells 21-7E in group 00, plane 00, row 00.

The three kinds of braces have the following meaning:

- Elements between curly brackets '{ }' indicate zero or more repetitions of those
elements.
- Elements between square brackets '[]' are optional elements.
- The parentheses '()' are used to denote groups of elements that are to be
regarded as a single element outside these parentheses.

A vertical bar '|' indicates a logical 'XOR', meaning that one and only one element on
either side of the bar can be selected. Multiple XOR pairs can be combined as is shown
by the definition of the identifier 'factor' in the above table.
Example of an application

```
Example of an application of the WSN-derived syntax notation.

| `person` | `name` | `(' yearofbirth ')` | `| 'childof' name` |
|----------|--------|---------------------|------------------|
|          | `name` | `string`            | `integer`        |

Assuming that `string` is a sequence of characters, and `integer` is an integer number, the following genealogical tree can be noted with the above syntax:

Josef Petrus van Leeuwen (1966)
  childof Rochus Antonius van Leeuwen
  and Anna Maria Daams

Rochus Antonius van Leeuwen (1925)
  childof Johannis Gerardus van Leeuwen
  and Anna Cornelia van Engelen

Anna Maria Daams (1930)
  childof Petrus Johannes Daams
  and Anna Maria Lentjes

Johannis Gerardus van Leeuwen (1894)

Anna Cornelia van Engelen (1889)

Petrus Johannes Daams (1898)

Anna Maria Lentjes (1900)
```
Appendix C

Textual and Graphical Notation of Feature Types and Feature Instances

This appendix summarises the syntax rules specifying the textual notation of Feature Types and Feature Instances, and the symbols that are used in the graphical notation for types and instances and for relationships. Both textual and graphical notation techniques are defined and discussed in chapter 7.
### Syntax of the classes of Feature Types

#### Syntax of the attribute `typeID` of the class `FeatureType`

| `typeID = sectionID '.' typeName` |
| `sectionID = libraryName '.' sectionName` |
| `libraryName = identifier` |
| `sectionName = identifier` |
| `typeName = identifier` |

#### Syntax for the attributes `typeCreated`, `typeAuthor`, and `typeDescr` of the class `FeatureType`

| `typeCreated = 'TypeCreated' '{' date '}'` |
| `typeAuthor = 'TypeAuthor' '{' string '}'` |
| `typeDescr = 'TypeDescr' '{' string '}'` |

#### Syntax for the attribute `typeBehaviour` of the class `FeatureType`

| `typeBehaviour = 'Behaviour' '{' { event-decl } '}'` |
| `event-decl = ( typeID | featureID ) roleName` |
| `eventID = identifier` |

#### General syntax for the definition of Feature Types

| `type-def = simple-type-def | enum-type-def | complex-type-def | geometric-type-def | constraint-type-def | handler-type-def` |

#### General syntax for the definition of Simple Feature Types

| `simple-type-def = string-type-def | integer-type-def | real-type-def | boolean-type-def` |

#### Syntax for the definition of string-based Simple Feature Types

| `string-type-def = 'string' typeID '{' string-body '}'` |
| `string-body = standard-body { string-domain-decl }` |
| `string-domain-decl = 'TypeDomain' '{' string-domain '}'` |
| `string-default-decl = 'TypeDefault' '{' string '}'` |
### Integer Feature Type

**Syntax for the definition of integer-based Simple Feature Types**

<table>
<thead>
<tr>
<th>Component</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer-type-def</td>
<td><code>integer-type-def = 'integer' typeID '{' integer-body ')'</code>.</td>
</tr>
<tr>
<td>Integer-body</td>
<td><code>integer-body = standard-body [ numeric-domain-decl ] [ integer-default-decl ] [ unit-decl ]</code>.</td>
</tr>
<tr>
<td>Numeric-domain-decl</td>
<td><code>numeric-domain-decl = 'TypeDomain' '{' numeric-domain ')'</code>.</td>
</tr>
<tr>
<td>Integer-default-decl</td>
<td><code>integer-default-decl = 'TypeDefualt' '{' integer ')'</code>.</td>
</tr>
<tr>
<td>Unit-decl</td>
<td><code>unit-decl = 'TypeUnit' '{' string ')'</code>.</td>
</tr>
</tbody>
</table>

### Real Feature Type

**Syntax for the definition of real-based Simple Feature Types**

<table>
<thead>
<tr>
<th>Component</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real-type-def</td>
<td><code>real-type-def = 'real' typeID '{' real-body ')'</code>.</td>
</tr>
<tr>
<td>Real-body</td>
<td><code>real-body = standard-body [ numeric-domain-decl ] [ real-default-decl ] [ unit-decl ]</code>.</td>
</tr>
<tr>
<td>Real-default-decl</td>
<td><code>real-default-decl = 'TypeDefualt' '{' real ')'</code>.</td>
</tr>
</tbody>
</table>

### Boolean Feature Type

**Syntax for the definition of boolean-based Simple Feature Types**

<table>
<thead>
<tr>
<th>Component</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean-type-def</td>
<td><code>boolean-type-def = 'boolean' typeID '{' boolean-body ')'</code>.</td>
</tr>
<tr>
<td>Boolean-domain-decl</td>
<td><code>boolean-domain-decl = 'TypeDomain' '{' boolean-domain ')'</code>.</td>
</tr>
<tr>
<td>Boolean-default-decl</td>
<td><code>boolean-default-decl = 'TypeDefualt' '{' boolean ')'</code>.</td>
</tr>
</tbody>
</table>

### Enum Feature Type

**Syntax for the definition of Enumeration Feature Types**

<table>
<thead>
<tr>
<th>Component</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enum-type-def</td>
<td><code>enum-type-def = 'enum' typeID '{' enum-body ')'</code>.</td>
</tr>
<tr>
<td>Enum-decl</td>
<td><code>enum-decl = 'TypeItems' '{' enum-items ')'</code>.</td>
</tr>
<tr>
<td>Enum-items</td>
<td><code>enum-items = identifier [',' identifier ]</code>.</td>
</tr>
<tr>
<td>Enum-domain-decl</td>
<td><code>enum-domain-decl = 'TypeDomain' '{' enum-domain ')'</code>.</td>
</tr>
<tr>
<td>Enum-default-decl</td>
<td><code>enum-default-decl = 'TypeDefualt' '{' identifier ')'</code>.</td>
</tr>
</tbody>
</table>
Appendix C

Syntax for the definition of Complex Feature Types

```plaintext
complex-type-def = 'complex' typeID [ '(' super-typeID ')' ]
                 '(' complex-body ')' .
super-typeID = typeID .
complex-body = standard-body [ complex-domain-decl ]
              [ complex-default-decl ]
complex-domain-decl = 'TypeDomain' '(' complex-domain ')' .
complex-default-decl = 'TypeDefault' '(' featureID ')' .
decomp-decl = 'Has' component-decl ; .
assoc-decl = 'Assoc' component-decl ; .
spec-decl = 'Spec' component-decl ; .
component-decl = ( type-component-decl | inst-component-decl ) .
type-component-decl = typeID roleName [ cardinality ]
                    [ '(' domain ')' ] [ param-list ]
                    [ '=' default ] .
inst-component-decl = roleName ['[ integer ]' '=?' featureID .
                    roleName = identifier .
cardinality = ['[ number ... ( number | '?' ) ]' .
domain = string-domain | numeric-domain | boolean-domain |
        enum-domain | complex-domain .
default = string | number | boolean | featureID .
```

<table>
<thead>
<tr>
<th>Specialisation</th>
<th>Decomposition</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Syntax for the declaration of parameter lists**

```plaintext
decl-param-list = param-decl { ',' param-decl } [ ',', ... ] .
param-decl = ( typeID | 'ANY' ) param-name .
param-name = identifier .
```

**Syntax for the definition of Geometric Feature Types**

```plaintext
geometric-type-def = 'geometry' typeID '(' [ decl-param-list ] ')' .
geometric-body = standard-body geometry-type .
geometry-type = 'TypeGeometry' '(' identifier ')' .
```

Modelling Architectural Design Information by Features
Appendix C

### Constraint Feature Type

**Syntax for the definition of Constraint Feature Types**

```
constraint-type-def = 'constraint' typeID '{' [ decl-param-list ] '}'
   '{' constraint-body '}' .
constraint-body = standard-body constraint-type .
constraint-type = 'TypeConstraint' '{' identifier '}' .
```

### Handler Feature Type

**Syntax for the definition of Handler Feature Types**

```
handler-type-def = 'handler' typeID '{' [ decl-param-list ] '}'
   '{' handler-body '}' .
handler-body = standard-body handler-procedure .
handler-procedure = 'Procedure' '{' code '}' .
```

**Syntax for domains**

```
string-domain = string ( ';' string ) .
numeric-domain = numeric-domain-term ( ';' numeric-domain-term ) .
numeric-domain-term = discrete-domain | continuous-domain | number .
discrete-domain = number ',' number '...' [ number ] .
continuous-domain = ( '<' | '[' ) ( '<' | number ) ',' ( '>' | ']' ) .
boolean-domain = boolean ( ';' boolean ) .
enum-domain = identifier ( ';' identifier ) .
complex-domain = featureID ( ';' featureID ) .
```

### Syntax of the classes of Feature Instances

**Syntax for the attributes featureID, created, author, descr, and behaviour of the class Feature**

```
featureID = [ sectionID '::' ] instanceName .
instanceName = identifier .
created = 'Date' '{' date '}' .
author = 'Author' '{' string '}' .
descr = 'Descr' '{' string '}' .
inst-behaviour = 'Behaviour' '{' { event-handler } '}' .
event-handler = featureID roleName '{' eventID '}' ';' .
```

**General syntax for the notation of Feature Instances**

```
feature-inst = simple-inst | enum-inst | complex-inst
  geometric-inst | constraint-inst | handler-inst .
standard-inst-body = created author descr
  inst-behaviour inst-relations .
```
### Syntax for the attribute instRelation of the class Feature

\[
\text{inst-relations} = \{ \text{inst-spec-decl} \mid \text{inst-decomp-decl} \mid \text{inst-assoc-decl} \}.
\]

- `inst-spec-decl` = 'Spec' inst-relation-decl
- `inst-decomp-decl` = 'Has' inst-relation-decl
- `inst-assoc-decl` = 'Assoc' inst-relation-decl
- `inst-relation-decl` = roleName [ ['integer'] ] = 'featureID' ;

### Syntax for the representation of Simple Feature Instances

\[
\text{simple-inst} = \text{typeID featureID} '=' \{ \text{simple-inst-body} \}.
\]

\[
\text{simple-inst-body} = \text{standard-inst-body} 'Value' \{ 'value' \}.
\]

- value = string | number | boolean

### Syntax for the representation of Enumeration Feature Instances

\[
\text{enum-inst} = \text{typeID featureID} '=' \{ \text{enum-inst-body} \}.
\]

\[
\text{enum-inst-body} = \text{standard-inst-body} 'EnumValue' \{ 'identifier' \}.
\]

### Syntax for the representation of Complex Feature Instances

\[
\text{complex-inst} = \text{typeID featureID} '=' \{ \text{complex-inst-body} \}.
\]

\[
\text{complex-inst-body} = \text{standard-inst-body} \{ \text{component-item} \}.
\]

- component-item = roleName [ ['integer'] ] = 'featureID' ;

### Syntax for the notation of parameter lists

\[
\text{param-list} = \text{parameter} \{ ',', \text{parameter} \}.
\]

\[
\text{parameter} = ( \text{featureID} \mid \text{SELF} \mid \text{NOTIFIER} ) \{ \text{component-param} \}.
\]

- component-param = ( '..' roleName [ ['integer'] ] )

### Syntax for the representation of Geometric Feature Instances

\[
\text{geometric-inst} = \text{typeID featureID} '\{ \mid \text{param-list} \}' '=' \{ \text{geometric-inst-body} \}.
\]

\[
\text{geometric-inst-body} = \text{standard-inst-body}.
\]
### Constraint Feature

<table>
<thead>
<tr>
<th>Syntax for the representation of Constraint Feature Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraint-inst = typeID featureID '{' [ param-list ] '}' '='</td>
</tr>
<tr>
<td>'{' constraint-inst-body '}' .</td>
</tr>
<tr>
<td>constraint-inst-body = standard-inst-body .</td>
</tr>
</tbody>
</table>

### Handler Feature

<table>
<thead>
<tr>
<th>Syntax for the representation of Handler Feature Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>handler-inst = typeID featureID [ '{' eventID ']' ]</td>
</tr>
<tr>
<td>'{' [ param-list ] '}' '='</td>
</tr>
<tr>
<td>'{' handler-inst-body '}' .</td>
</tr>
<tr>
<td>handler-inst-body = standard-inst-body .</td>
</tr>
</tbody>
</table>


Cross, N. 1984, Developments in Design Methodology. Chichester: Wiley & Sons Ltd.

Modelling Architectural Design Information by Features.


Modelling Architectural Design Information by Features


Modelling Architectural Design Information by Features


Modelling Architectural Design Information by Features
Literature References


Architectural design, like many other human activities, benefits more and more from the ongoing development of information and communication technologies. The traditional paper documents for the representation and communication of design are now replaced by digital media. CAD systems have replaced the drawing board and knowledge systems are used to integrate expert knowledge in the design process. Product modelling is one of the most promising approaches in the developments of the last two decades, aiming in the architectural context at the representation and communication of the information related to a building in all its aspects and during its complete life-cycle.

However, after studying both the characteristics of the product modelling approach and the characteristics of architectural design, it is concluded in this research project that product modelling does not suffice for support of architectural design. Architectural design is characterised mainly as a problem solving process, involving ill-defined problems that require a very dynamic way of dealing with information that concerns both the problem and emerging solutions. Furthermore, architectural design is in many ways an evolutionary process. In short term this is because of the incremental approach to problem solving in design projects; and in long term because of the stylistic development of designers and the continuous developments in the building and construction industry in general. The requirements that are posed by architectural design are concentrated in the keywords extensibility and flexibility of the design information.
models. Extensibility means that designers can extend conceptual models with definitions that best suit the design concepts they wish to utilise. Flexibility means that information in design models can be structured in a way that accurately represents the design rationale. This includes the modelling of incidental characteristics and relationships of the entities in the model that are not necessarily predefined in a conceptual model.

In general, product modelling does not adequately support this dynamic nature of design. Therefore, this research project has studied the concepts developed in the technology of Feature-based modelling, which originates from the area of mechanical engineering. These concepts include the usage of Features as the primitives for defining and reasoning about a product. Features have an autonomous function in the information model, which, as a result, constitutes a flexible network of relationships between Features that are established during the design process. The definition of Features can be specified by designers to formalise new design concepts. This allows the design tools to be adapted to the specific needs of the individual designer, enlarging the library of available resources for design.

In addition to these key-concepts in Feature-based modelling as it is developed in the mechanical engineering context, the project has determined the following principles for a Feature-based approach in the architectural context. Features in mechanical engineering are used mainly to describe the lowest level of detail in a product's design, namely the characteristics of its parts. In architecture the design process does not normally follow a strictly hierarchical approach and therefore requires that the building be modelled as a whole. This implies that multiple levels of abstraction are modelled and that Features are used to describe information at the various abstraction levels. Furthermore, architectural design involves concepts that are non-physical as well as physical; Features are to be used for modelling both kinds. The term Feature is defined in this research project to reflect the above key-concepts for this modelling approach:

A Feature is an autonomous, coherent collection of information, with semantic meaning to a designer and possibly emerging during design, that is defined to formalise a design concept at any level of abstraction, either physical or non-physical, as part of a building model.

Feature models are built up entirely of Features and are structured in the form of a directed graph. The nodes in the graph are the Features, whereas the arcs are the relationships between the Features. Features can be of user-defined types and incidental relationships can be added that are not defined at the typological level.

An inventory in this project of what kind of information is involved in the practice of modelling architectural design is based on the analysis of a selection of sources of architectural design information. This inventory is deepened by a case study and results in the proposition of a categorisation of architectural Feature types.
The major part of the research project concerns the definition of the so-called FBM framework, an infrastructure for Feature-based modelling for the support of architectural design. The framework defines the process of formalisation of design concepts into Feature type definitions and the process of creating Feature models consisting of instances of Feature Types. This framework is based on three abstraction layers of information definition. The middle layer contains the definitions of types of Features; the bottom layer contains the actual Feature data, Feature Instances of these types representing particular design cases; the top layer is the so-called meta-layer that defines the format for both Feature Types and Feature Instances. It is the top layer that enables the extensibility and flexibility of the Feature-based information structures. This layer consists of six corresponding pairs of classes for Feature Types and Feature Instances. The formal definition of these classes is presented and both a textual and graphical notation technique are provided.

After the presentation of the FBM framework, a number of implementation issues are discussed. This includes various strategies for the formalisation of design concepts into Feature Type definitions and several scenarios for the creation of Feature models. The aspect of dynamic manipulation of both types and models is discussed, as well as the possible contribution of the developed approach to efforts in standardisation and communication of data models. Aspects concerning the presentation of Feature related data in a virtual design environment are discussed. The role of the FBM framework in the developments of the VR-DIS research programme at Eindhoven University of Technology is explained. Within this research programme, a sequence of prototypes are developed for implementation of the FBM framework, forming the basis for a design support system. The second of these prototypes forms the basis for a case study on the application of the Feature-based modelling approach in architectural design.

The FBM framework for modelling architectural design information fulfils the requirements posed by the dynamic nature of architectural design on extensibility and flexibility of information models for design support. The implementation of this framework and the application in an architectural design case study have demonstrated that this approach can successfully be developed as a basis for design support systems that support rather than obstruct creativity.
Samenvatting

Het bouwkundig ontwerpen, zoals vele andere menselijke activiteiten, ervaart steeds meer voordelen van de voortschrijdende ontwikkeling van informatie en communicatie technologieën. De traditionele papierteksten voor de representatie en communicatie van een ontwerp zijn vervangen door digitale media. CAD systemen hebben het tekenbord vervangen en expert systemen worden gebruikt om kennis te integreren in het ontwerpproces. Product modelleren is een van de meest veelbelovende benaderingen in de ontwikkelingen van de laatste twee decennia, met als doelstelling in de bouwkundige context het representeren en communiceren van de informatie die is gerelateerd aan een gebouw in al zijn aspecten en gedurende zijn gehele levenscyclus.

Na bestudering, echter, van zowel de karakteristieken van het product modelleren als van het bouwkundig ontwerpen, wordt geconcludeerd in dit onderzoeksproject dat product modelleren niet voldoet voor de ondersteuning van het bouwkundig ontwerpen. Bouwkundig ontwerpen karakteriseert zich vooral als een probleemoplossend proces, waarbij slechts gedefinieerde problemen aan de orde zijn die vereisen dat op zeer dynamische wijze met informatie wordt omgegaan; informatie welke naast het probleem ook de oplossingen behelst zoals die zich aandienen. Bovendien is bouwkundig ontwerpen op velelei wijze een proces van evolutie. Op de korte termijn is er de incrementele benadering waarmee ontwerpproblemen worden opgelost; op de langere termijn vertonen ontwerpers een ontwikkeling in hun ontwerpstijl en zijn er ook steeds doorgaande ontwikkelingen in de bouwsector in het algemeen. De eisen die het bouwkundig ontwerpen oplegt kunnen worden samengevat met de sleutelwoorden uithoudbaarheid en flexibiliteit van ontwerpinformatie modellen. Uithoudbaarheid betekent dat ontwerpers conceptuele modellen kunnen uitbreiden met definities die het best de ontwerpconcepten weergeven welke zij wensen te gebruiken. Flexibiliteit betekent dat informatie in ontwerpmodellen kan worden gestructureerd op een manier die een accurate representatie vormt van de ontwerpgedachte. Dit houdt in het modelleren van incidentele eigenschappen en relaties van entiteiten in het model die niet noodzakelijk zijn voor gedefinieerd in een conceptueel model.

In het algemeen ondersteunt product modelleren niet afdoende dit dynamische karakter van het ontwerpen. Dit onderzoeksproject heeft derhalve de concepten bestudeerd die zijn ontwikkeld in de technologie Feature-based modelling. Deze stamt uit het domein van de werktuigbouwkunde. Hierin worden Features gebruikt als de primitieven voor het definieren en redeneren over een product. Features hebben een autonome functie in het informatie model dat als gevolg daarvan bestaat uit een flexibel netwerk van relaties tussen Features die worden vastgelegd gedurende het ontwerpproces. De definitie van Features kan door ontwerpers worden bepaald voor het formaliseren van nieuwe ontwerpconcepten. Op die manier kunnen ontwerpgerelateerde eigenschappen worden aangepast aan de specifieke behoeften van de individuele ontwerper en kan de verzameling van beschikbare hulpmiddelen voor het ontwerpen worden vergroot.

Modelling Architectural Design Information by Features
Benevens deze kernconcepten in Feature-based modelling, zoals ontwikkeld in de
context van de werktuigbouwkunde, zijn in dit project de volgende additionele principles
vastgesteld voor de Feature gebaseerde benadering in bouwkundige context. In de
werktuigbouwkunde worden Features vooral gebruikt op het meest gedetailleerde niveau
van productontwikkeling, namelijk voor de eigenschappen van de productonderdelen. In
de bouwkunde heeft het ontwerpproces normaliter geen hierarchische benadering en
vereist derhalve dat het gebouw als geheel wordt gemodelleerd. Dit impliceert dat
meerdere abstractieniveaus worden gemodelleerd en dat Features worden gebruikt voor
het beschrijven van informatie op de diverse abstractieniveaus. Daarnaast spelen in het
bouwkundig ontwerpen zowel tastbare als niet-tastbare concepten een rol; Features
moeten daarom beide soorten concepten representeren. Om bovenstaande principles van
deze modellerwijze weer te geven is de term Feature in dit onderzoeksproject als volgt
gedefinieerd:

Een Feature is een autonome, samenhangende verzameling van informatie,
met semantische betekenis voor een ontwerper en zich mogelijk vormend
tijdens het ontwerpen, die wordt gedefinieerd om een ontwerp concept te
formaliseren op enig abstractieniveau, tastbaar of niet-tastbaar, als deel van
een gebouwmodel.

Feature modellen zijn geheel opgebouwd uit Features en gestructureerd in de
vorm van gerichte grafen. De knopen in de graaf zijn de Features, de lijnen zijn de
relaties tussen de Features. Features kunnen van een type zijn dat door de gebruiker is
gedefinieerd en incidentele relaties die niet op typologisch niveau zijn gedefinieerd
can worden toegevoegd.

Een inventarisatie in dit project van de soorten informatie die zich in de praktijk
van het bouwkundig informatie modelleren voordoen is gebaseerd op de analyse van een
selectie van bronnen van bouwkundige ontwerp informatie. Deze inventarisatie wordt
verdiept middels een case study en resulteert in een voorstel voor de categorisatie van
bouwkundige Feature types.

Het grootste gedeelte van dit onderzoeksproject omvat de definitie van het zogenoemde
FBM framework, een infrastructuur voor Feature-based modelling voor de
ondersteuning van het bouwkundig ontwerpen. Het framework definiert het proces van
formalisatie van ontwerpconcepten tot Feature types en het proces van het creëren van
Feature modellen welke bestaan uit instanties van Feature types. Dit framework is
gedefinieerd op drie abstractieniveaus voor informatie definitie. Het middenste niveau bevat
de definities van de Feature Types; het onderste niveau bevat de daadwerkelijk Feature
data, Feature Instanties die ontwerpprojecten representeren en welke zijn gedefinieerd
door de Feature Types; het bovenste niveau is het zogenoemde meta-niveau dat het
formaat definiert voor zowel Feature Types als Feature Instanties. Dit bovenste niveau
bewerkstelligt de uitbreidbaarheid en flexibiliteit van de Feature gebaseerde informatie
structuren. Dit niveau bestaat uit zes corresponderende tweetallen van klassen voor
Feature Types en Feature Instanties. De formele definitie van deze klassen wordt gegeven en in zowel een tekstuele als grafische notatie techniek is voorzien.

Na de presentatie van het FBM framework worden een aantal implementatie issues behandeld. Hieronder vallen verschillende strategieën voor de formalisering van ontwerpconcepten tot Feature Type definities en een aantal scenario's voor het creëren van Feature modellen. Het aspect van het dynamisch manipuleren van zowel types als modellen wordt bediscussieerd, alsook de mogelijke bijdrage van de ontwikkelde methode aan het gebied van standaardisatie en communicatie van data modellen. Aspecten met betrekking tot de presentatie van Feature gerelateerde data in een virtuele ontwerp omgeving worden besproken en de rol van het FBM framework in de ontwikkelingen van het VR-DIS onderzoeksprogramma van de Technische Universiteit Eindhoven wordt toegelicht. Binnen dit onderzoeksprogramma is een reeks van prototypen ontwikkeld voor de implementatie van het FBM framework, welke de basis vormen voor een ontwerpondersteunend systeem. Het tweede prototype uit deze reeks is gebruikt in een case study over de toepassing van de Feature-based modelling benadering in het bouwkundig ontwerpen.

Het FBM framework voor het modelleren van bouwkundige ontwerpinfoformatie vervult de eisen die werden gesteld door het dynamische karakter van het bouwkundig ontwerpen aan de uitbreidbaarheid en flexibiliteit van ontwerp ondersteunende informatiemodellen. De implementatie van dit framework en de toepassing daarvan in een case study hebben aangetoond dat deze benadering succesvol kan worden ontwikkeld als basis voor ontwerpondersteunende systemen die creativiteit ondersteunen, niet belemmeren.
Index

Subject index

A
abstraction mechanisms, 38
aggregation, 39
API, 209, 212, 213
application, 213
architectural information, 10, 14, 26, 86
in product models, 93
inventory, 85
inventory case study, 98, 108, 110
modelling, See information modelling
sources, 86
aspects, 10, 11, 33
association, 39, 125, 141, 166, 170, 171,
177, 184, 190
Association, 144
B
BAS-CAAD, 40, 53, 54, 77
BCCM, 41, 42, 202
behaviour, 38, 55, 78, 131, 135, 155, 161,
162, 188, 190, 192
C
CAD, 17, 18, 20, 32, 33, 42, 46, 47, 53, 54,
76, 202
case study, 98, 110, 213
classification, 17, 87, 91, 99, 121, 225
3D, 19
strategies, 182
COMBINE, See IDM
communication, 13, 20, 201
ComplexFeature, 165
ComplexFeatureType, 140
complexity, 15, 79, 80, 86
component, 166
component based approach, 24

Modelling Architectural Design Information by Features
Component, 141
concept, See design concepts
corporal information model, 48
conclusions, 229
constraint solving, 70, 149, 186, 209
ConstraintFeature, 170
ConstraintFeatureType, 149
core-model, 33. See also BCCM
D
data-exchange
approaches, 34
decision support system, 21, 22
DeclaredParameterList, 146
decomposition, 39, 78, 79, 83, 105, 125, 141, 166, 177, 184, 190
Decomposition, 144
defaults, 155
design
actions, 225
architectural design information, 26
case study, 213
complexity, 15
concept formalisation, See design concepts
design by Features, 67
dynamic data models, 53
dynamic nature of, 46
enhanced process, 13
ill-defined problems, 14
information modelling, 45
management, 14
product modelling, 44
design concepts
abstract, non-physical, 79
formalisation, 75, 81, 82, 180, 183
identification, 180
design methodology, 181
design support system, 52
discussion, 234
documentation, 91, 101
documents
digital, 16
domains, 155
E
EDM, 40, 46, 54, 78
Elementenmethode '91, 17, 37, 108
EnumerationFeature, 165
EnumerationFeatureType, 139
event handling, 151, 197
expert systems, 21
EXPRESS, 41
EXPRESS-G, 41, 237
extensibility, 49
in the FBM framework, 120
F
FBM framework, 119
case study, 213
extensibility, 120
Feature instances, 160
Feature types, 130
flexibility, 120
implementation, 179. See also prototypes
information structure, 123
notation, 131, 247
processes, 121
prototypes, 209
twofold instantiation, 129
typology of relationships, 125
FBM in architecture, 73
additional principles, 80
analogy problems, 78
concepts, 74
conclusions, 229
definition of term Feature, 82
Feature recognition, 198
Feature types, See Feature types framework, See FBM framework
processes, 121
FBM in mechanical engineering, 59
actual research, 69
approaches, 66
background, 60
concepts, 64
design by Features, 67
Feature conversion, 68
Feature recognition, 66
taxonomies, 62
Feature: See also FBM
definition of the term in architecture, 82
definition of the term in mechanical engineering, 60
recognition, 198
Feature instances
base class, 160
classes, 160
ComplexFeature, 165
ConstraintFeature, 170
EnumerationFeature, 165
GeometricFeature, 169
HandlerFeature, 171
incident relationships, 175
interface, 204
models, 174
notation, 131, 247
SimpleFeature, 163
subclasses, 162
Feature instantiation, 122, 195, 234
defaults, 196
handled, 197
incompleteness, 196
macros and templates, 197
recognition, 198
scenarios, 196
Feature model, 174
manipulation, 199
structure, 83
Feature type definition, 121, 180, 183, 234
Feature types
base class, 133
categories in architecture, 109, 113
classes, 130
ComplexFeatureType, 140
ConstraintFeatureType, 149
domains and defaults, 155
EnumerationFeatureType, 139
Generic, 115, 123, 124, 201, 234
Generic versus Specific, 115, 124
GeometricFeatureType, 147
HandlerFeatureType, 151
in architecture, 85
interface, 204
libraries, 158
notation, 131, 247
SimpleFeatureType, 136
Specific, 123, 124
standardisation, 124, 201, 234
subclasses, 135
Feature-based modelling, See FBM Features

Index

case study, 213
in architecture, 75, 81, 213
flexibility, 50
in the FBM framework, 120
formalisation, See design concepts
framework, See FBM framework
future work, 234
G
GARM, 26, 40, 75, 76
generalisation, 38
Generic Feature Types, See Feature types,
Generic
geometric models, 18
advantages and disadvantages, 18
linked databases, 20
parametric, 19
procedural programming, 18
GeometricFeature, 169
GeometricFeatureType, 147
H
HandlerFeature, 171
HandlerFeatureType, 151
I
ICIS, 88, 100, 108
IDM, 26, 93, 106, 109
IFC
comparison to, 202
implementation, 179. See also prototypes
VR-DIS, 207
information
architectural, See architectural information
architectural design, 26
implicit, 14
integration of, 16
quality of, 11
state of, 11
subjects of, 10
types of, 11
information modelling
approaches, 16
developments, 9
objectives, 12
inheritance, 39, 78, 125, 140, 166, 183, 194

Modelling Architectural Design Information by Features
Index

instantiation, 67, 69, 70, 128, 129, 152, 155. See also Feature instantiation
integration, 16, 30
interface, 204
ISO, 18, 30, 36, 37, 40, 41, 47, 88, 130, 132, 202, 238, 244

K
knowledge
  design domain, 180
  integration in modelling systems, 20
  knowledge-based systems, 21, 22, 23

M
macro, 197
mapping, 35, 202
modification, 122

N
NBD, 91, 93, 103
NL-SfB, 17, 87, 89, 91, 93, 99
notation, 131, 247

O
object orientation, 23, 32, 120, 125, 129
  classification strategies, 182
  concepts, 38
  constraint solving, 149, 209
  ObjectStore, 213
  prototyping, 209
  object-type, 38

P
parameter, 143
Parameter, 168
presentation, 12
product modelling, 25, 29, 30. See also SME. See also FDM. See also BAS CAAD.
  See also IDM. See also GARM
  approaches, 34
  architectural design, 44
  architectural information in, 93
  background, 31
  concepts, 33
  developments in B&C, 40
  Features for AFC, 77
  objectives, 32
  standardisation, 40
  prototypes, 209

Q
quality control, 13
questions
  research, 27, 230

R
reasoning, 12
recognition
  Feature recognition, 66, 198
  relationships, 125, 194. See also abstraction mechanisms
  association, See association
  decomposition, See decomposition
  incidental, 121, 128, 175
  inheritance, See inheritance
  instance level, 121, 128, 175
  role names, 194
  specialisation, See specialisation
  specification, See specification
  standardisation, 194
representation, 12, 181
research
  questions, 27, 230
rule-based, 21

S
shape grammar, 21
SimpleFeature, 163
SimpleFeatureType, 136
SME, 54, 78
specialisation, 38, 78, 124, 125, 141, 144, 145, 166, 177, 194
Specific Feature Types, See Feature types, Specific
  specification, 114, 126, 141, 177, 184, 190
  Specification, 144
STABU, 17, 87
standardisation, 40, 201
STEP, 36, 77, 124
  comparison to, 201
  discussions within, 43
  EXPRESS, 41
  EXPRESS-G, 41, 237
  introduction, 40
  mapping, 202
  sub-model, 33.

Modelling Architectural Design Information by Features
T
  template, 197
V
  views, 10, 33
  vocabulary, 180

Author index
Achten, H.H., 4, 181, 182, 214, 225, 226, 227
ACM, 243
Alexander, C., 47
Allen, J.F., 149
Amor, R.W., 33, 37, 202
Augenbroe, G.L.M., 26, 40, 95
Autodesk, 30
Beitz, W., 181
Bidarra, R., 71
Björk, B.-C., 18, 40, 93, 94, 95, 105, 109
Booch, G., 23, 183
Bronsvoort, W.F., 61, 63, 65
Chamberlain, M.A., 62
Chang, G.J., 61
Chen, S.S., 24
Chen, Y.-M., 69
Chermayeff, S., 47
Chopra, D., 57
Clayton, M.J., 54, 78
Coomans, M.K.D., 205, 206, 207
Coyne, R.D., 47, 50
Cross, N., 14, 47
Crowley, A., 40
Cunningham, J.J., 61
DeMartino, T., 69
Dissel, D.J. van, 26
Dixon, J.R., 61
Dohmen, M., 149
Duan, W., 70
Dubois, A.M., 40
Eastman, C.M., 32, 33, 40, 46, 54, 78
Eekels, J., 181
Ekholm, A., 40, 53, 77
Emmerik, M.J.G.M. van, 61, 62, 63
Faux, I.D., 61
Flemming, U., 21
French, M.J., 181
Fridqvist, S., 40, 53, 77
VR-DIS, 3, 6, 204, 207, 209, 225, 227, 233, 235
W
  WSN, 132, 243

Gero, J.S., 22
Gielingh, W.F., 26, 40, 75
Haas, W., 17, 43
Hall, A.D., 181
Han, J., 69
Hannus, M., 35
Harfmann, A.C., 24
Henderson, M.R., 61
Hoffmann, C.M., 60, 69, 70
Hosking, J.G., 33, 37, 202
ISO TC184, 37, 41, 47, 130, 202, 238
ISO TC59, 30, 88
Jansen, F.W., 61, 63
Jellema, R., 15
Jeng, T.S., 54, 78
Jessurun, A.J., 212
Joan-Arinyo, R., 69, 70
Juan, R., 60
Karstula, K., 35
Kelleners, R.H.M.C., 149, 209
Kraker, K.J. de, 68
Krishnamurti, R., 21
Laakko, T., 69
Lawson, B., 46, 76, 79
Leeuwen, J.P. van, 21, 43, 63, 80, 122, 214, 225, 226, 227
Liebich, T., 37, 42, 202, 203
Lucardie, G.L., 23
Luiten, G.T., 25, 30, 31
Maher, M.L., 21, 22
Mäntylä, M., 60, 61, 63, 64, 65, 66, 69
Markus, T.A., 46
Martin, D., 43
Martin, J., 23, 126, 127
Maver, T.W., 46
McCullough, M., 21
Meeran, S., 62
Merendonk, P.H. van, 26
Meyer, B., 183

Modelling Architectural Design Information by Features
Curriculum Vitae

Jos van Leeuwen, born February 6, 1966 in Hedel, completed secondary school (Gymnasium B) in June 1984 at the Jeroen Bosch college in 's-Hertogenbosch. In June 1988 he received an engineers degree (ing.) at the Higher Technical School in 's-Hertogenbosch, graduating with a project titled ‘Guidelines for the design and construction of BMW Dealer-facilities’. After some months’ employment at an architect’s office, he fulfilled military service from September 1988 until November 1989.

In September 1989 he recommenced studies at the Eindhoven University of Technology, Faculty of Architecture, Building, and Planning. He graduated January 1993 at the Calibre / Building Information Technology group on the development of an environment, integrated in a CAD system, for the definition and application of design-rules. During his study he was a trainee at Abacus, Strathclyde University in Glasgow, on CAD development and design visualisation. Also during his study, from August 1990 to November 1993, he was employed part-time at A CAD-Consult BV, 's-Hertogenbosch, initially as CAD developer, later as project manager. In January 1993 he started his PhD study at Eindhoven University of Technology, being employed as research fellow (AiO) at the Calibre / Building Information Technology group. In this group he was involved in several research projects in areas of product modelling, design support, and CAD-, DBMS-, and Internet-development. Additionally, he took part in lecturing in these areas, in the undergraduate as well as post-graduate curriculum. Since September 1997 he continues these activities as Assistant Professor in the Design Systems group where his main tasks lie within the VR-DIS research programme and the education in the Building Information Technology curriculum.
BOUWSTENEN is een publikatierreeks van de Faculteit Bouwkunde, Technische Universiteit Eindhoven. Zij presenteert resultaten van onderzoek en andere activiteiten op het vakgebied der Bouwkunde, uitgevoerd in het kader van deze Faculteit.

BOUWSTENEN zijn verkrijgbaar bij:

Publikatiewinkel 'Legenda'
Hoofdgebouw 4.92
Faculteit Bouwkunde
Technische Universiteit Eindhoven
Postbus 513
5600 MB Eindhoven

of telefonisch te bestellen:
040 - 2472293
040 - 2472529

Kernredaktie
Prof. dr dipl. ing. H. Fassbinder
Prof. dr R. Oxman
Prof. ir H.H. Snijder
Prof. dr H.J.P. Timmermans
Prof. ir J.A. Wisse

International Advisory Board

Prof. ir N.J. Habraken
Massachusetts Institute of Technology
Cambridge U.S.A.

Prof. H. Harms
Technische Universität Hamburg
Hamburg, Duitsland

Prof. dr G. Helmberg
Universität Innsbruck
Innsbruck, Oostenrijk

Prof. dr H. Hens
Katholieke Universiteit Leuven
Leuven, Belgie

Dr M. Smets
Katholieke Universiteit Leuven
Leuven, Belgie

Prof. dr F.H. Wittmann
ETH - Zürich
Zürich, Zwitserland
Reeds verschenen in de serie
BOUWSTENEN
	nr.1
Elan, a computermodel for building energy design, theory and validation
M.H. de Wit
H.H. Driessen
R.M.M. van der Velden
	nr.2
Kwaliteit, keuzevrijheid en kosten
Evaluatie van experiment Klarendal, Arnhem
drs J. Smeets
C. le Nobel, arch. HBO
M. Broos, J. Frenken, A. v.d. Sanden
	nr.3
Crooswijk
van 'bijzonder' naar 'gewoon'
drs V. Smit
ir K. Noort
	nr.4
Staal in de woningbouw
ir E.J.F. Delsing
	nr.5
Mathematical theory of stressed skin action in profiled sheeting with various edge conditions
ir A.W.A.M.J. v.d. Bogaard
	nr.6
Hoe berekenbaar en betrouwbaar is de coëfficiënt k in \(x\)-ko en \(x\)-ks?
ir K.B. Lub
drs A.J. Bosch
	nr.7
Het typologisch gereedschap
Een verkennende studie omtrent typologie en omtrent de aankoop typologisch onderzoek
J.H. Luiten arch. HBO
	nr.8
Informatievoorziening en beheerprocessen
ir A. Nauta / drs J. Smeets (red.)
Prof. H. Fassbinder (projectleider)
ir A. Proveniers,
drs J.v.d. Moosdijk
	nr.9
Strukturering en verwerking van tijdgegevens voor de uitvoering van bouwwerken
ir W.F. Schaefer
ir P.A. Erkelens
	nr.10
Stedebouw en de vorming van een speciale wetenschap
K. Doevendans
	nr.11
Informatica en ondersteuning van ruimtelijke besluitvorming
dr G.G. van der Meulen
	nr.12
Staal in de woningbouw, korrosiebescherming van de begane grondvloer
ir E.J.F. Delsing
	nr.13
Een thermisch model voor de berekening van staalplaatbetonvloeren onder brandomstandigheden
ir A.F. Hamerlinck
	nr.14
De wijkgedachte in Nederland Gemenschapsstreven in een stedebouwkundige context
dr ir K. Doevendans
dr R. Stolzenburg
	nr.15
Diaphragm effect of trapezoidally profiled steel sheets.
Experimental research into the influence of force application
ir A.W.A.M.W. v.d. Bogaard
	nr.16
Versterken met spuit-ferrocement.
Het mechanische gedrag van met spuit-ferrocement versterkte gewapende betonbalken
ir K.B. Lub
ir M.C.G. van Wanroy
nr.17
De tractaten van
Jean Nicolas Louis Durand
ir G. van Zeyl

nr.18
Wonen onder een plat dak.
Drie opstellen over enkele vooronderstellingen van de stedebouw
ir K. Doevendans

nr.19
Supporting decision making processes
A graphical and interactive analysis of multivariate data
drs W. Adams

nr.20
Self-help building productivity
A method for improving house building by low-income groups
applied to Kenya 1990-2000
ir P. A. Erkelens

nr.21
De verdeling van woningen:
een kwestie van onderhandelen
drs V. Smit

nr.22
Flexibiliteit en kosten in het ontwerp
-proces Een besluitvormingondersteunend model
ir M. Prins

nr.23
Spontane nederzettingen begeleid
Voorwaarden en criteria in Sri Lanka
ir P.H. Thung

nr.24
Fundamentals of the design of bamboo structures
O. Arce-Villalobos

nr.25
Concepten van de bouwkunde
Prof. dr ir M.F.Th. Bax (red.)
dr ir H.M.G.J. Trum (red.)

nr.26
Meaning of the site
Xiaodong Li

nr.27
Het woonmilieu op begrip gebracht
Jaap Ketelaar

nr.28
Urban environment in developing countries
editors: dr ir Peter A. Erkelens
dr George G. van der Meulen

nr.29
Strategische plannen voor de stad
Onderzoek en planning in drie steden
Prof. dr H. Fassbinder (red.)
ir H. Rikhof (red.)

nr.30
Stedebouwkunde en stadsbestuur
ir Piet Beekman

nr.31
De architectuur van Djenné
Een onderzoek naar de historische stad
P.C.M. Maas

nr.32
Conjoint experiments and retail planning
Harmen Oppewal

nr.33
Strukturformen Indonesischer Bautechnik Entwicklung methodischer Grundlagen für eine 'konstruktive pattern language' in Indonesien
Heinz Frick

nr.34
Styles of architectural designing
Empirical research on working styles and personality dispositions
Anton P.M. van Bakel

nr.35
Conjoint choice models for urban tourism planning and marketing
Benedict Dellaert

nr.36
Stedelijke Planvorming als co-produktie
Prof. dr H. Fassbinder (red.)
nr 37
Design Research in the Netherlands
editors: Prof. dr. R.M. Oxman,
Prof. dr ir. M.F. Th. Bax,
Ir H.H. Achten

nr 38
Communication in the Building Industry
Bauke de Vries

nr 39
Optimaal dimensioneren van gelaste plaatliggers

nr 40
Huisvesting en overwinning van armoede
dr.ir. P.H. Thung en dr.ir. P. Beekman (red.)

nr 41
Urban Habitat: The environment of tomorrow
George G. van der Meulen,
Peter A. Erkelens

nr 42
A typology of joints
John C.M. Olie

nr 43
Modeling constraints-based choices for leisure mobility planning
Marcus P. Stemerding

nr 44
Activity-based travel demand modeling
D. Ettema

nr 45
Wind-induced pressure fluctuations on building facades
Chris Geurts

nr 46
Generic Representations
Henri Achten

nr 47
Johann Santini Aichel
Dirk De Meyer

nr 48
Concrete behaviour in multiaxial compression
Erik van Geel

nr 49
Modelling site selection
Frank Witlox

nr 50
Ecolemma model
Ferdinand Beetstra

nr 51
Conjoint approaches to developing activity-based models
Donggen Wang

nr 52
On the effectiveness of ventilation
Ad Roos

nr 53
Conjoint modeling approaches for residential group preferences
Eric Molin
Propositions
accompanying the thesis

Modelling Architectural Design
Information by Features

Jos P. van Leeuwen
29 Juni 1999
The confusion that exists in the 'faculteit Bouwkunde' of Eindhoven University of Technology in translating the English term 'Architecture' to either the Dutch 'Architectuur' or 'Bouwkunde' is not due to a misinterpretation of these terms, but to an unresolved discussion on the changing role of architects in current and future design and construction processes. (this thesis, p. 26)

Data-explosions are not caused by the implementation of information technologies; information technologies are the first attempts to actually tackle the problem of data-explosion that already exists in design. (this thesis, p. 15)

The moment that my thesis advisor awards me with a new title, he himself loses one. Although this sad fact should not be a reason for any further delay of this promotion, it is a consequence that better not follows by rule.

As long as computer aided design tools cannot be adapted to the demands of the individual designer, they will be experienced as an obstruction for creativity.

When some lecturers in this faculty do not allow the usage of computers in design projects, with all students dragging one on their back, this does not imply anything about computer aided design, but rather about computer fearing assessment.

The way of thinking, in western medicine, about alternative medicine often does the term alternative much injustice.

In many of the health problems in our present time, western medicine is better regarded as the last alternative.

We must regard the architectonic space as an addition to the space of nature, by which the conflict between this space and our experience-space is removed. (Dom Hans van der Laan in De architectonische ruimte, p. 12, transl. JPvL)

Disregarding their appreciation of the style determined for publications of the faculty of Architecture, doctoral students should be obliged to publish their thesis in the series of Bouwstenen.

By way of the legislation concerning the granting of residential permits, the Dutch government shows to have more confidence in the relationship between employer and employee than in the relationship between husband and wife.
De verwarring die bestaat in de faculteit Bouwkunde van de Technische Universiteit Eindhoven bij de vertaling van de Engelse term 'Architecture' naar het Nederlandse 'Architectuur' of 'Bouwkunde' is niet te wijten aan een misinterpretatie van deze termen, maar aan een onopgeloste discussie over de veranderende rol van de architect in huidige en toekomstige ontwerp- en bouwprocessen. (dit proefschrift, p. 26)

Data-explosies worden niet veroorzaakt door de toepassing van informatie technologieën; informatie technologieën zijn de eerste pogingen om het probleem van data-explosie dat al in het ontwerpen bestaat daadwerkelijk aan te pakken. (dit proefschrift, p. 15)

Het moment dat mijn promotor mij promoveert ontvang ik een nieuwe titel, terwijl hij er zelf een verliest. Hoewel deze trieste gebeurtenis geen reden voor verder uitstel van deze promotie mag zijn, is dit een consequentie die toch beter niet als regel volgt.

Zolang computer ondersteunde ontwerpgereedschappen niet kunnen worden aangepast aan de wensen van de individuele ontwerper, zullen zij worden ervaren als een belemmering voor de creativiteit.

Wanneer sommige docenten in deze faculteit niet toestaan dat de computer wordt gebruikt in ontwerpprojecten, terwijl alle studenten er een op hun rug mee sjouwen, zegt dat niet zozeer iets over het computer ondersteund ontwerpen, dan wel over het computer bevrizzend beoordeeld.

De wijze waarop in de westere geneeskunde vaak wordt gedacht over alternatieve geneeswijzen doet de term alternatief groot onrecht aan.

Bij veel hedendaagse gezondheidsproblemen wordt de westere geneeskunde beter als laatste alternatief beschouwd.

De architectonische ruimte moeten wij beschouwen als een toevoegsel aan de natuurlijke ruimte, waardoor het conflict tussen deze ruimte en onze ervaringsruimte wordt opgeheven. (Dom Hans van der Laan in De architectonische ruimte, p. 12)

Ongeacht hun oordeel over de huisstijl die is vastgesteld voor publicaties van de faculteit Bouwkunde zouden promovendi verplicht moeten worden hun proefschrift uit te geven in de reeks Bouwstenen.

Via de wetgeving met betrekking tot het verlenen van verblijfsvergunningen toont de Nederlandse overheid dat zij meer vertrouwen heeft in de relatie tussen werkgever en werknemer, dan in de relatie tussen twee echtgenoten.
Modelling Architectural Design Information by Features

Architectural design information is modelled for a number of reasons, such as formal representation of the design, enhanced communication between participants, and support of the design task itself. For the latter, it is especially important that design information is modelled in a way that closely corresponds to the intentions of the individual designer. The information model must exhibit sufficient flexibility and extensibility to evolve with design as it proceeds.

The research project presented in this thesis develops a framework for information modelling that builds on the concepts of Feature-based modelling. This framework offers the capabilities required for supporting the dynamic ways information is dealt with during the architectural design process. It allows designers to formally define the concepts that emerge during design and to add incidental characteristics and relationships to the entities in a model, even if these are not typologically defined. This approach leads to the modelling of design information that closely reflects the design rationale and offers strong potentials for the development of true design support systems.

ISBN 90-6814-554-1