TECHNISCHE
UNIVERSITÄT
DRESDEN

Fakultät
Bauingenieurwesen

A Mapping Language for Concurrent Engineering Processes

Peter Katranuschkov

Herausgeber:
Prof. Dr.-Ing. R. J. Scherer

P. Katranuschkov   A Mapping Language for CE Processes

Schriftenreihe
des Lehrstuhls für Computeranwendung im Bauwesen
Heft 1

CIB

1

ISBN 3-86005-280-2

A Mapping Language for Concurrent Engineering Processes

Peter Katranuschkov

Schriftenreihe des Lehrstuhls für Computeranwendung im Bauwesen
Herausgeber: Univ.-Prof. Dr.-Ing. R. J. Scherer

ISBN 3-86005-280-2

Lehrstuhl für Computeranwendung im Bauwesen, TU Dresden

Postanschrift	Besucheranschrift
Technische Universität Dresden 01062 Dresden	Nürnberger Str. 31a 2. OG, Raum Nr. 204 01187 Dresden

Tel.	+49 351/463-2966
Fax:	+49 351/463-3975
E-Mail:	scherer@cib.bau.tu-dresden.de
WWW:	http://cib.bau.tu-dresden.de

Fakultät Bauingenieurwesen
Institut für Baumechanik und Bauinformatik

TECHNISCHE UNIVERSITÄT DRESDEN

Lehrstuhl für Computeranwendung im Bauwesen
Univ.-Prof. Dr.-Ing. Raimar J. Scherer

Diese Arbeit wurde unter dem Titel:

A Mapping Language for Concurrent Engineering Processes

« Eine Interoperabilitätsmethode zur Unterstützung
von Concurrent-Engineering-Prozessen »

an der Fakultät Bauingenieurwesen der Technischen Universität Dresden als

DISSERTATION

von Peter Katranuschkov
geboren am 22. Juni 1953 in Sofia / Bulgarien

zur Erlangung des akademischen Grades eines Doktor-Ingenieurs (Dr.-Ing.)
genehmigt.

Gutachter:

Hauptreferent:	Prof. Dr.-Ing. Raimar J. Scherer, Technische Universität Dresden
Korreferenten:	Prof. Dr. James H. Garrett, Jr., Carnegie Mellon University, Pittsburgh
	Prof. Dr.-Ing. Peer Haller, Technische Universität Dresden

Tag der öffentlichen Verteidigung: 10. November 2000

For my family

Preface

When I started work on this text I was well aware that its size would eventually become considerably larger than it is usual, and that it was going to be focused on two (and not one) major topics, covered in *different breadth and depth*. Nevertheless, I decided not to change the initially outlined structure of the presentation. In the following I will briefly explain why it is as it is.

The study reported in the thesis is the result of my research activities at the Dresden University of Technology in the period 1996-2000, conducted in conjunction with my participation in the international research and development project ToCEE sponsored by the European Union. However, many of the initial ideas that led to this study emerged already in 1994-1995, during my work on an earlier European project, COMBI, at the University of Karlsruhe. COMBI was one of the first projects in the AEC domain that dealt with the development of an information system for cooperative design on the basis of product data technology and it was at that time – after doing a lot of modelling and having experienced several "integration" and "islands of automation" problems on our own shoulders – that my advisor, Professor R. J. Scherer, and I started to think of a conceptual solution that can help glue together the inherently multidiscipline non harmonised model world of CAE/CIC.

Coupled with the specific requirements of concurrent engineering in building construction which were broadly investigated in the frames of the ToCEE project, this gave birth to the main goal of my research efforts: *the conceptualisation and prototype implementation of an interoperability approach for a distributed client-server environment for concurrent engineering processes based on product data technology*.

The core of this approach consists of a suggested new method for context-independent model mapping built upon the definition of a novel *mapping language,* CSML.

CSML presents an extensive and formal means by which the model transformations needed for the successful information exchange and sharing between different discipline domains can be declaratively expressed and then automatically executed as to map the data specified by one model schema to their respective representation specified by another model schema. As such, CSML is the main contribution of this thesis. It provides a basis for tackling the semantic interoperability problems in a non harmonised or only partially harmonised modelling environment, which is of quite general applicability, and *not* specifically dedicated to the solution of certain concurrent engineering tasks. For example, CSML can be used to facilitate the integration of application tools into a modelling environment for which they were not originally designed, it can help in creating adequate translators enabling the transformation of project data from one large model, such as IFC, to another large model as e.g. the CIMsteel Logical Product Model, or it can assist in the development of domain extension models within the IFC framework by alleviating the requirements for strict, tight harmonisation with the components of the kernel model.

However, for several reasons the emphasis of CSML *is* on concurrent engineering.

First, many of the requirements of concurrent engineering to an information system can strongly influence the design and the implementation of the mapping language and methods. Such requirements include the ability to represent partial or incomplete mappings, the tackling of concurrent operations on one and the same data set, the support of cooperation and negotiation processes by enabling the work on discipline-specific model views individually and in parallel, the existence of already populated models as target of a mapping task etc.

On the other hand, the adequacy of the mapping language to deal with the abovementioned issues is of utmost importance to the basic conceptual idea that, to support concurrency, it is not necessary for all models maintained in the computer system to be fully harmonised, and the shared data do not have to be entirely and continuously consistent. This data could only be coordinated on an as-needed basis, at specific points in time, allowing to perform all concurrent engineering processes in a more natural way.

At last, to facilitate the integration of design applications by means of unified programming interfaces, it is essential to treat the different interoperability aspects of the environment in their inter-relationships. In other words, the mapping language and methods need to be incorporated as part of a consistent overall representation approach.

Thus, many conceptual decisions with respect to model mapping can be inferred from the superior and much broader subject of concurrent engineering, and many decisions with respect to the overall environment are in turn dependent on the developed mapping approach. To help understand these decisions better, I have deliberately chosen to precede the core of the presentation by an extended overview of the background, the architecture and the operability of an envisaged environment in which model mapping can efficiently be applied. This resulted in three more or less distinct parts of the text as described below.

Organisation of the thesis

The thesis is divided into nine chapters along with seven appendices and a reference section.

Its first part, dedicated to the *overall concept of a concurrent engineering environment*, covers the first four chapters. It is broader in scope but with less discussion about why certain choices are made and from what selection of alternatives.

Chapter 1 introduces the specific motivation and objectives of the study, the adopted research method and the demonstration examples pulled up for the validation of the developed concepts.

Chapter 2 presents the background of the proposed solution approach, focusing, on one hand, on the principles of concurrent engineering methodology, and on the other hand, on the state of the art of IT application in the building construction sector and the state of research and development in engineering data modelling.

Chapter 3 outlines the basic concepts of the suggested environment and identifies the interoperability issues that have been the main target of the research effort.

Chapter 4 presents an approach for solving fundamental systemic interoperability problems in the proposed environment. It is at the same time a transition between the first and the second part of the thesis as it goes into more detail than the preceding three chapters to present a logically consistent conceptual solution based on the idea of information containers, overlaid by a generalised communication model and object-oriented project data operations, enhanced by server-side knowledge-based functionality. However, in order to remain in scope, in-depth discussions are kept concise here as well.

The second part, comprising chapters 5 to 7, is dedicated to the main *semantic interoperability problems that have to be dealt with in a distributed, heterogeneous model world.* These problems are considered as the hardest and most challenging task in the context of the thesis. In contrast to the first part, the presentation here is more focused and goes into considerably more detail. It includes a comprehensive discussion of the problem domain, related efforts, alternatives to the proposed new solution approach, detailed description of the developed model mapping language, a (heuristic) validation of the developed concepts and a critical appraisal of the achieved results.

Chapter 5 discusses the general concepts and the state of research in the domain, and outlines the proposed solution.

Chapter 6 introduces the developed mapping language, CSML. Along with a formal and comprehensive presentation of the syntax and semantics of CSML, it provides also a detailed proof of concept on the basis of a wide range of typical mapping exercises.

Chapter 7 describes three larger case studies, providing evidence for the field applicability of the suggested approach.

The third part, comprising chapters 8 and 9, is dedicated to the practical *validation and evaluation* of the developed concepts. For this purpose, a prototype project data management server and selected sample clients have been implemented as part of the research work.

Chapter 8 outlines the main features of the prototyped server, PROMISE, and presents a more comprehensive implementation example, adapted from the ToCEE project, which covers most major aspects of the developed approach.

Chapter 9 provides an overall evaluation of the concepts and the prototype implementation, and outlines envisioned areas for future research.

Technical details and background information related to the work are presented in the seven appendices complementing the thesis.

Acknowledgements

This thesis would not have been possible without the kind support of many persons who helped in many different ways to make it happen.

First and foremost, I would like to thank my advisor, Professor Raimar J. Scherer, for all fruitful discussions, his guidance and encouragement, his enthusiasm for the ideas encompassed in this thesis, and his continuous efforts in funding rounds.

My thanks go also to the co-reviewers, Professor James Garrett Jr. from the Carnegie Mellon University in Pittsburgh and Professor Peer Haller from the Dresden University of Technology, for all the precious time invested in reading and reviewing the thesis, and for providing me with a number of valuable comments and suggestions for improvement of the text.

I thank my wife Veronika for her love and patience, and all the support and understanding throughout these years. I thank also our two lovely kids for their understanding. You did not have much of a father in periods of hard work, but now it is your turn and I can get my own back!

I thank my mother and my father for their trust in me, their love, and their continuous encouragement.

I thank all the staff of the chair of Applied Computer Science in Civil Engineering at the Dresden University of Technology for their support and assistance. During all these years I enjoyed a nice atmosphere, filled with many fruitful discussions over a cup of coffee (or occasionally, a beer). My special thanks go to Markus Hauser and Rainer Wasserfuhr who were always there with a friendly advice, an interesting suggestion, or, even more valuable, critical comments. Special thanks also to Ilona Jantzen who helped me more than she probably thinks to get this thesis out the door.

I thank Christoph Meinecke and Mohammad Mehrafza for the stimulating discussions during my employment at the University of Karlsruhe. Due to an awful accident Mohammad will not be able to read these lines any more. I will never forget him!

Financial support for this work was provided through my employment at the Dresden University of Technology, and especially through the financing of the European projects COMBI and ToCEE by the European Union and the project partners. This support is gratefully acknowledged. The hard work in these two projects was accompanied by many lively conversations and “after hours” discussions which proved to be of immense value to my work. Special thanks to Byron Protopsaltis, Edith Ammermann, Juha Hyvärinen, Matti Hannus, Mike Clift, Richard Junge, Robert Amor, Tero Hemiö, Thomas Liebich, Žiga Turk, and all the others of the COMBI and the ToCEE group. You were all good friends and a great help!

At last, I would like to thank also Bo-Christer Björk, Charles Eastman, James Garrett Jr., Richard Wittenoom, Robert Logcher and Steve Fenves. Although some of them do not even know me, their impressive ideas often provided that little help needed to understand things better.

Thanks to everybody who provided direct and indirect assistance in the preparation of this thesis!

Contents

Chapter 1: Introduction

"Begin at the beginning", the King said, very gravely,
"and go on till you come to the end: then stop."
– L. Carroll, Alice in Wonderland

The rapid globalisation of the world markets in the last decade has brought forth new challenges to the competitiveness of the building industry, strongly influencing the organisation of design and construction work. Along with the increased realisation of construction projects in virtual enterprises comprised of physically distributed specialist teams, a growing interest in the introduction of advanced production methodologies and the use of innovative information technology (IT) solutions can be widely observed. Today, it is recognised that concurrent engineering is the production methodology that can bring greatest competitive advantage to a company, *if appropriately applied*.

This thesis presents a novel approach for computer-supported concurrent engineering processes on the basis of product data technology and with special emphasis on major interoperability problems that need to be dealt with. Chapter 1 sets the stage by introducing the specific motivation and objectives of the work, the adopted research method and the demonstration examples pulled up for the validation of the developed concepts.

1.1 Motivation

According to (Scherer 1998a), in the present time the major features of concurrent engineering are: (1) the collaborative work of physically distributed teams, (2) simultaneous engineering, i.e. taking into account the full product life cycle during the design stage by means of forecasting and simulation, and (3) the synchronised cooperative engineering of one and the same part of the product by different experts. These features can be realised by an IT framework and a virtual product model that allows different data views but includes also *"functionalities for data management, such as transformations, consistency checking, monitoring, control and notifications* […]".

This view is now broadly recognised. However, in spite of a general understanding of the benefits of concurrent engineering and the rapid hardware and software development, extending the areas of automation far beyond CAD and numerical analysis tools, the state of IT solutions for concurrent engineering is not yet satisfactory.

On the one hand, this is certainly due to some differences between building construction and other industries that have negative influence on IT development. In (IAI 1999a) these differences are outlined as follows:

- other industries have higher profits enabling them to invest larger sums in IT development;
- other industries have few, very large, key organisations who can drive a technology to suit their requirements;
- other industries have less complex supply and communication chains.

Additional reasons are often seen in the fragmented and multidiscipline nature of the building industry, and the one-off products it has to deal with.

On the other hand, although broadly discussed, the principles of concurrent engineering and their implications to information technology in building construction are not yet well understood. The few commercially available systems claiming to support concurrent engineering mainly mimic traditional document-oriented methods of work. Being document, and not model based, they fail to provide many vital features, such as comprehensive change and conflict management, system-wide data consistency, coordination of alternative solutions. As a result, with such systems only a limited increase in productivity can be achieved.

By examining the state of computing in the AEC domain, many researchers have consequently come to the conclusion that the major problems inhibiting the successful application of IT lie with the conceptual models of buildings (cf. Junge 1991; Björk 1992; Björk 1994; Hannus et al. 1995b; Wix 1996; Amor 1997). The understanding that the lack of formal, standardised, comprehensive models limits both the capabilities and the integration of CAD applications, as well as the successful information exchange and sharing in construction projects, has lead to the inauguration of a number of research projects tackling different portions of the commonly known as “islands of automation” problem. However, in spite of all recent achievements, essential concurrent engineering issues, such as the management of the process, product, documentation and communication flows, are still being handled in a fragmented manner, as individual, mutually independent, or at best only partially inter-connected systems. Thus, to date, the major open problems related to the realisation of a fully interoperable, computer-supported concurrent engineering environment are not related to the components, but to the *conceptual modelling of the environment itself*. Whilst in the last years a number of comprehensive product, process and document models have been proposed, a clear concept for a modelling environment that enables their integration, and at the same time fulfils the requirements of concurrent engineering, is not yet available. Many aspects related to the *interoperability* of the environment, such as the communication between users and applications in a distributed, heterogeneous system, the semantic integration of the implemented data models, and the functionality of the system in reaction to data conflicts and modifications, are not yet sufficiently investigated, and are solved only through the application of ad hoc developed tools and methods.

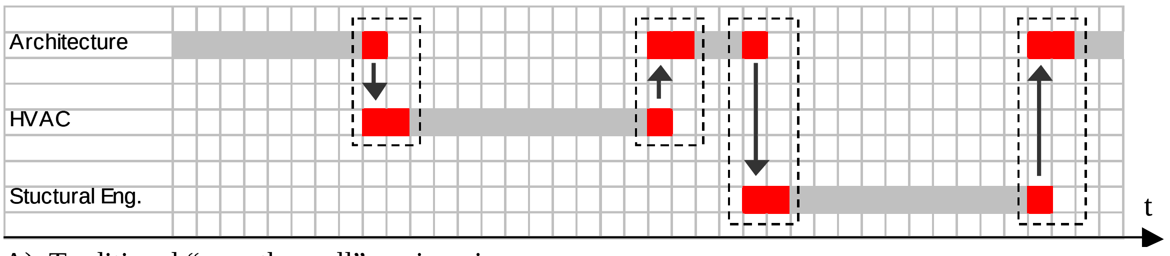
To be more explicit, let us consider an example situation:

> Mr. Archibald is a highly qualified architect, in charge of an architectural design office. Well aware that today’s design work is unthinkable without adequate computer support, he has equipped his team with newest architectural CAD software, along with several office automation and groupware tools. His employees work on fully networked, Internet-enabled workstations, and they are all skilled computer users. However, in his latest project Mr. Archibald was again disappointed by his “computer investments”. After finishing the scheme design, he transmitted the prepared data via Internet first to the office of his fellow services engineer, Mr. Hatchwack, expecting that there might be some minor problems with the HVAC system that should be duly checked. But having waited for several days for a response, he discovered that he was faced with much more serious problems than initially assumed. First, it turned out that Mr. Hatchwack is using a different CAD system, and he had to spend many work hours merely to “feed” the data of Mr. Archibald into it. Second, he was not able to fulfil the HVAC design without moving some elements around, and thus had to create an alternative solution, producing an entire new set of drawing plans. Third, not all proposed changes could be seen on the drawings which required a bulky memorandum to explain.
>
> Many days spent on similar activities by one of Mr. Archibald’s employees, and a number of open questions needing additional requests for information and negotiations, brought the carefully planned work schedule at rock bottom, the structural engineer was kept off work, and even worse, some problems that arose later with the bearing structure had to be solved in wild haste, at the expense of space design, in order to finish in time. The final design solution was not one to be proud of, and many minor defects had to be fixed during construction, resulting in additional unplanned costs.

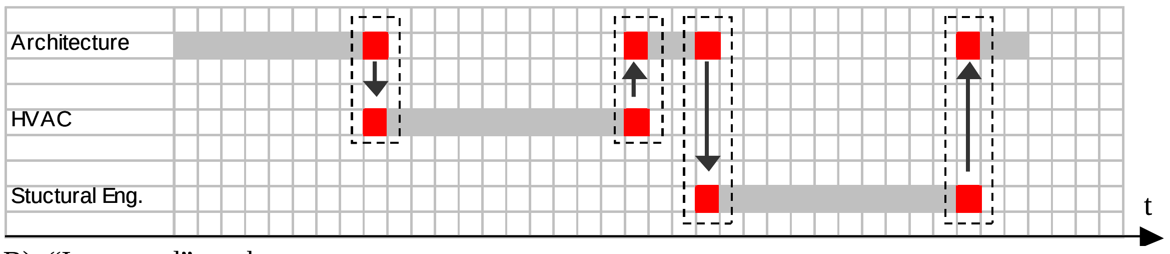
This simple example is freely invented, yet not that far from reality. Indeed, by using advanced design-support tools each designer can considerably improve his own professional work, but there is very little he can do to coordinate his work with others, to think of cooperative solutions in which multidiscipline design problems can be better balanced, and to minimise waiting times. Thus, even though advanced design tools can make individual activities more effective compared to manual work, the overall process remains basically sequential, leaving little room for cooperative design decisions and reduction of lead time. The result is an improved version of “over-the-wall” design practices, far from the vision of concurrent engineering.

More efficient cooperation can be achieved through the use of compatible, i.e. *integrated* computer-aided design systems. In fact, such arrangements are made often before the beginning of large construction projects. However, concurrent engineering requires more than that. Integrated software solutions allowing seamless information exchange can considerably cut coordination times and tedious, non value generating processes, but taken alone they cannot contribute much to the effective organisation of team work and truly collaborative project realisation.

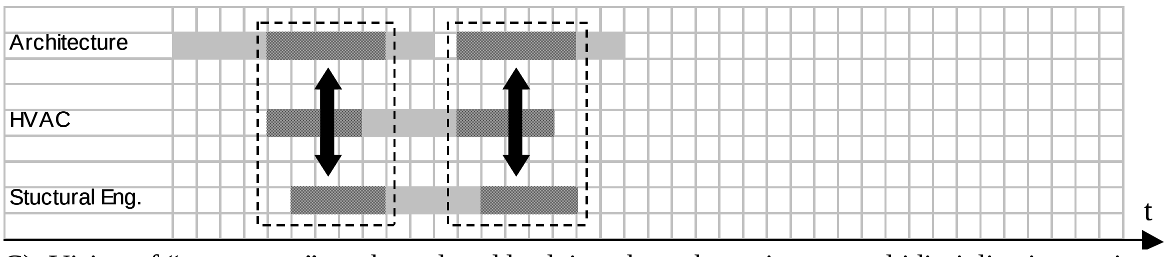
This aspect becomes more obvious by examining fig. 1.1, showing three “possible” time schedules for the above example.



A) Traditional “over the wall” engineering
$T_{arch} = 18$, $T_{HVAC} = 13$, $T_{struct.} = 13$, $T_{total} = 40$, Resources = 44 time units.



B) “Integrated” work
$T_{arch} = 16$, $T_{HVAC} = 12$, $T_{struct.} = 12$, $T_{total} = 36$ (10% gain), Resources = 40 (9% gain).



C) Vision of “concurrent” work: reduced lead time through continuous multidiscipline interaction
$T_{arch} = 18$, $T_{HVAC} = 12$, $T_{struct.} = 12$, $T_{total} = 19$ (> 50% time gain), Resources = 42 (5% gain).

Fig. 1.1: Time schedules of the design process by different project realisation approaches

The presented time data on this figure are fictitious, but in correspondence to each other, and in line with the generally accepted proportions of creative and non value generating work, shown by light grey and dark boxes respectively. In the last diagram, coordinated activities are presented by checked boxes, denoting that coordination and communication in the concurrent engineering approach should happen hand in hand with "real" design work. However, while the given time schedules are fictitious, the portions of non value generating coordination activities, ranging between 15 and 30 percent of the total time, are not. Thus, by considering the design process as a whole, from the perspective of time, the great potential of continuously practised concurrent work can readily be seen (cf. Smith & Reinertsen 1991) [*)].

On the other hand, fig. 1.1 shows only idealised time models: the last diagram presents a future vision, and the current situation is not that bleak. In fact, as indicated in (Schulz 1996), long before "virtual enterprise" and "concurrent engineering" became established terms, design and construction projects have been performed by teams of small specialist companies that cooperate together in often transient relationships to apply their skills and expertise on a particular project.

Whenever possible, such team work has always been performed *concurrently*, leaving the impression that concurrent engineering is the natural way the building industry operates, and that there is *not much to learn* from other industries with respect to concurrent engineering methodology, but there is *much to gain* by adopting off-the-shelf information technology solutions to support the concurrent engineering process.

Unfortunately, due to several factors, this is not quite true.

The first key point here is in the words "**whenever possible**".

In general, concurrent work is possible when each team member knows what to do, when to do it, how to do it, what do others do, who is dependent on his/her actions, and whose other actions influence his/her work. However, to know what to do and when to do it requires to be able to take initiative proactively, being aware of the overall goals and the overall processes comprising a construction project. To know how to do some piece of work requires not only appropriate knowledge and expertise in the own domain but also awareness of the inter-dependencies with other domains and time schedules, and an understanding of the consequences of a specific design solution to the overall performance characteristics of the designed construction facility. To know these inter-dependencies requires awareness of the responsibilities, the competence and the tasks of the other team members, and to know what others do means to have the right information at the right time.

It is easy to see that many of these activities are related to coordination and communication, "swallowing" a large amount of the available resources[**)]. Thus, <u>the first challenge</u> is to find IT methods to improve this "whenever possible", which is of great importance both to the duration and cost of a project, and, indirectly, to the quality of the product itself.

The second key point is in the word "**concurrently**".

Integrated information processing can improve coordination and communication times, but substantial productivity gains can only be achieved by simultaneously carried out tasks.

---

*) Further aspects enabled by concurrent engineering that are not shown on fig. 1.1 include better design solutions through enhanced coordination, early simulation for detection of risks and prevention of wrong decisions, improved quality control.

**) In (IAI 1999a) it is mentioned that "[...] *up to 30% of the cost of a building project is due to the fractured processes and communication of the AEC/FM industry*".

This presumes local, independent work which inevitably leads to diverging design solutions and data conflicts. Therefore, to be able to work "concurrently" it is necessary to ensure data consistency and convergence of the individually designed building systems (spaces and space boundaries, bearing structure, HVAC, electrical installations etc.), and at the same time enable as much as possible autonomous, non coordinated, creative work.

This dichotomy cannot be solved only by integrating design applications and shared database access to common project information. Besides, additional problems can arise when an IT system is not only logically, but also "geographically" distributed, which is a common situation in building construction. Thus, <u>the second challenge</u> is to find information technology methods that would allow to merge diverging design solutions *when the necessity arises*, without imposing rigid coordination mechanisms inhibiting creative work.

The third key point is in the **relationship to other industries**.

On the one hand, due to its natural characteristics (one-off products, highly fragmented infrastructure), the building industry has developed an intuitive concurrent engineering approach long before any manufacturing industry. However, this approach is still largely intuitive. The theoretical methodology of concurrent engineering has been created by other industries and is seldom properly applied in building construction practice. As a consequence, its implications to building IT are weakly studied and insufficiently considered.

On the other hand, off-the-shelf IT solutions available from other industries can offer only partial support because: (1) they are designed to act in other environments, with different work processes and information needs, and (2) they are mostly huge, expensive systems developed primarily for the needs of key organisations in powerful industry sectors (automotive, aerospace), which makes them inadequate to the modest resources available in the AEC domain. Although in a slightly different area, the limited success of STEP in building construction provides sufficient evidence in that respect.

What remains is to rely on internal efforts. However, because of the limited resources for research and development, it is imperative to coordinate as much as possible development work. Thus, <u>the third challenge</u> is to find an approach for efficient IT support for concurrent engineering which is tightly aligned with existing standardisation efforts. Currently, these are the Industry Foundation Classes developed by the IAI.

The last point refers to the **target of the research effort**.

From the preceding discussion it is obvious that the work presented in this thesis addresses the design phase of a construction project. Of course, concurrent engineering methodology does not apply only to design, and some considerations regarding the construction phase will be given later on in chapter 2. However, as markedly pointed out in (Wittenoom 1998) "*the most effective time to optimise project effectiveness and profitability is during the early stages of model realization. This stage accounts for a small proportion of the development budget but decisions taken will have major impact on life cycle effectiveness, capital expenditure, and operating costs. It is important to get these decisions right as soon as possible*". I will only add "with the help of an IT framework".

1.2 Research Objectives

The overall **goal** of this research is the development of a model-based approach for a distributed IT environment enabling adequate consideration of the major requirements of the concurrent engineering methodology. The specific **focus** is on the interoperability of the environment and its framework and not on the elaboration of single components like product and process models, design tools, or user interfaces.

The developed approach should support the parallel, independent work of the members of a distributed design team, as well as a broad spectrum of coordination and cooperation activities. It should provide for the integration of different software applications used for different domain tasks, and it should have sufficient representational power to support the evolution and the concretisation of design information from initial sketches and ideas to final ready-to-build specifications. Last but not least, the developed concepts should be aligned with existing standardisation work, taking into account the requirements, constraints and limitations of relevant IT and building standards to guarantee the practical value of the achieved results.

Depending on the initial hypotheses, different design strategies can be applied to the solution of the problems related to the conceptualisation of an environment that can satisfy the above broadly defined goal.

The approach proposed in this thesis is based upon the following set of inter-related **hypotheses**:

1. A main prerequisite for the envisaged environment is an underlying logically consistent object-oriented modelling framework enabling model-based project realisation, as opposed to existing document-oriented approaches mimicking traditional paper-based work. This framework should be *implementable*, i.e. it should represent not only the components of the designed product and the related design/production processes, but should encompass the components of the IT system as well.

2. The core of the modelling framework should be built upon a *standardised shared project model*, designed in accordance with the principles of product data technology. However, this model cannot be expected to include all information needed in all subtrades of building design and construction. Along with it, other *domain models* may be needed, and these models may not be harmonised, neither horizontally, i.e. with each other, nor vertically, i.e. with the shared project model itself.

3. To allow for the consideration of alternative solutions, developed in parallel by one or more design professionals, it should be possible for each designer to work with his own discipline-specific models and application tools *locally* and *independently*, in his own private workspace. Hence, a distributed, a priori non coordinated model world has to be taken into account.

4. As individual design decisions may reference or influence the model data of others, adequate methods ensuring the overall consistency of the information have to be provided. My hypothesis here is that, instead of trying to impose continuous consistency by constantly updating the content of the shared project model at each single action, these methods should guarantee the consistency of the data as needed by the designers themselves, at discrete coordination points. This means that the freedom to experiment and the coordination of design decisions should be left solely to the human designers. The system should support, and not dictate the organisation of the work.

5. The last hypothesis is that, taken alone, the commonly used object-oriented modelling paradigm is not always sufficient to describe the full required functionality of a concurrent engineering environment, and that industry standard concepts of object-oriented programming, such as inheritance, encapsulation and polymorphism, are not always the best choice for the realisation of this functionality. Where appropriate, more advanced knowledge-based concepts enhancing standard object technology need to be considered.

In accordance with that, the following **objectives** have been set up (listed in the order observed by the elaboration and the respective presentation of the developed approach):

1. Provide principal concepts for a distributed client/server system for concurrent engineering in building design, with emphasis on the architecture and the formal specification of interoperability models, components and services;

2. Use product data technology as baseline by aligning the developed concepts as far as possible with STEP methodology;

3. Use the IFC models as general reference models of the proposed approach by coordinating all specific components for concurrent engineering support with the IFC modelling architecture;

4. Study possibilities to incorporate advanced knowledge-based features into the concurrent engineering environment;

5. Develop an interoperability approach to enable the solution of problems related to the concurrent, multidiscipline work on multiple, non harmonised domain models;

6. Implement a prototype software system as proof of the developed concepts.

The most challenging of these objectives are the first, the fourth and especially the fifth, whereas the second and the third have served basically as guidance for several conceptual decisions, and the last is specifically dedicated to the validation of the developed approach.

1.3 Related Research

As a methodology covering the full life cycle of product development and maintenance, concurrent engineering has many facets. A number of its aspects have been individually investigated in one form or another in the last decade. Currently, there exists considerable know-how related to various components of the envisaged concurrent engineering IT environment in the areas of conceptual product and process modelling, database management, distributed processing, computer networks and communication technology. Valuable results have been achieved in a series of academic studies and in a number of large research and development projects, as well as in the frames of major commercial systems.

Impressive research efforts in the area of Building Construction have been undertaken in the European projects ATLAS (Böhms & Storer 1994; Tolman & Poyet 1995), CIMSTEEL (Watson & Crowley 1995), COMBI (Scherer 1995; Scherer & Sparacello 1996), COMBINE (Augenbroe 1995a), PISA (Braun et al. 1994), VEGA (Amar et al. 1997) and ToCEE (Amor et al. 1997; Scherer 2000); the US projects SEED (Flemming & Woodbury 1995; Fenves et al. 1995) and DICE (Peña-Mora et al. 1995; Sriram & Logcher 1996); the German project "A4 – digitales Bauen" (Hovestadt 1993); the Finnish OOCAD/RATAS approach (Serén et al. 1993; Hannus et al. 1995a); the international research initiative IRMA (Luiten et al. 1993); the Australian "Realization Model" (Wittenoom 1997) etc. More recent efforts include the projects CICC, CONCUR, COMMIT and DESCRIBE (cf. Kamara et al. 2000), and the CLDC framework (conceptual framework for Concurrent Lifecycle Design & Construction).

Intensive research and development work has been conducted in other industry branches as well, most notably in the aerospace, mechanical, petrotechnical, shipbuilding, high-tech and defense industries. Outstanding examples provide the large European projects RISESTEP /in the aerospace domain/ (Figay 1998), REMAP /mechanical industry and robotics/ (Sieberer & Keber 1998), EPISTLE (Angus & Dziulka 1998) and PIPPIN (Thomson 1998) /petro-technical and process plant facilities/, carried out in conjunction with the POSC/CAESAR project, and MARITIME /shipbuilding/ (de Brujin et al. 1995), along with a cluster of follow-up projects (MARVELOUS, MAREXPO, OPTIMISE etc.) and in close cooperation with the US Navy Niddesc project. There are also a number of industry led efforts as e.g. the US NIIIP project /data protocols for high performance computing networks for the automotive, aerospace and defense industries/ funded by DARPA (Hardwick et al. 1997) and the OMG's PDM Enablers initiative (Waskiewicz & Siegel 1996), as well as proprietary PDM solutions, such as METAPHASE software (SDRC 2000), widely used by major companies in the automotive, aerospace, high-tech and defense industries.

Much standardisation work has been performed in the frames of CALS, ISO, IAI and WfMC that can usefully be applied. Of particular interest to this study are the integration initiatives undertaken in the ISO STEP standard and by the IAI.

The objective of STEP (ISO 10303-1 1994) is the sharing of unambiguous models between applications through standardisation of application-independent data models, called “application protocols” (APs). These models are expected to provide the link between the dedicated domains of the individual applications and that of STEP-conformant information, enabling both error-free data exchange and the use of shared data repositories. Currently this approach is being successfully applied in some subdomains of the automotive, process plant and shipbuilding industries. However, because of the complicated and resource intensive standardisation process, the acceptance of STEP in building construction is still relatively low.

In contrast to that is the work of the International Alliance for Interoperability (IAI 1999a). Targeting an evolutionary model with releases currently on bi-annual basis, it enjoys wide support by over 600 member organisations. The goal of the IAI is to define and promote a “universal building language”, the IFCs, as a basis for sharing AEC project information globally, across disciplines and technical applications, and through the product life cycle.

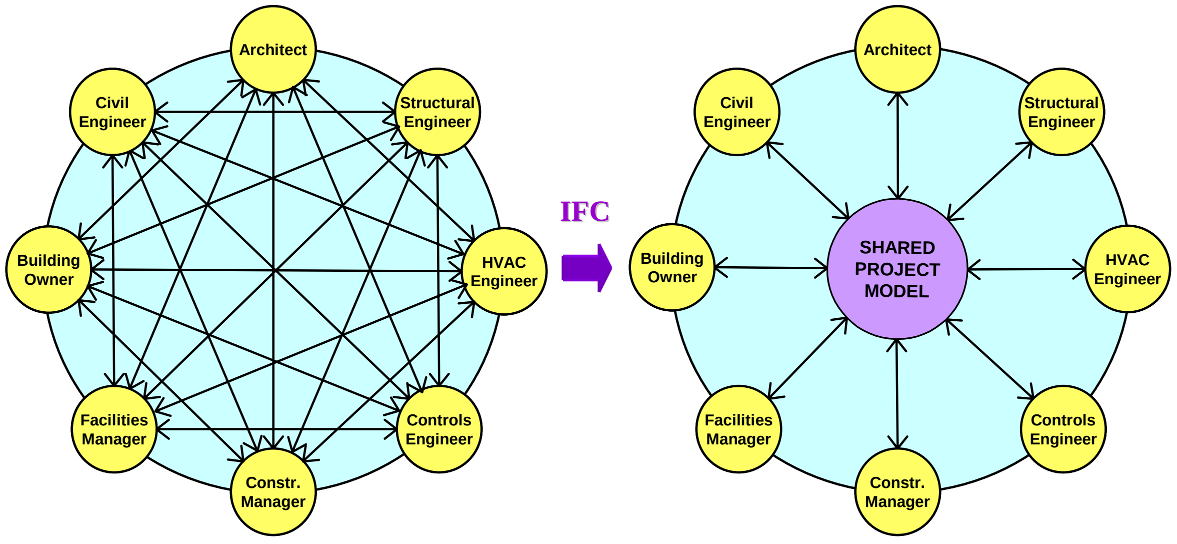


Fig. 1.2: The IAI vision of interoperability / adapted from (IAI 1999a) /

The IAI vision, expressed schematically on fig. 1.2 above, is to achieve interoperability through a shared project model to which all kinds of applications can interface. New generation software incorporating the IFC project model specifications is thus expected to bring at least the following two benefits to the way a project team works together: (1) coordinating information, and (2) transferring the information intact (Herold 1997).

Many of these efforts are referenced as appropriate in the following chapters. However, both STEP and IAI focus mainly on software integration, envisaging, in the medium term, a shared project data repository as a remedy to all "islands of automation" problems, whereas many issues related to the interoperability of implemented IT environments are overlooked or, at best, treated on theoretical level, without much consideration w.r.t. their practical validation and scalability. The majority of the academic studies propose solutions only for specific difficult problems in narrow domains, such as model evolution and multiple design views, conflict resolution methods on the basis of constraint satisfaction, Internet-enabled communication and negotiation methods etc., paying little attention to the alignment of the developed methods in a coherent concurrent engineering approach. Larger projects that have investigated the construction of product data based environments in a wider scope have been primarily concerned with data modelling and integration aspects, ignoring other substantial requirements related to the architecture and the operability of a run-time environment. For example:

- the ATLAS project developed a very broad layered modelling framework, with considerable influence on the later IFC architecture, but did barely elaborate detailed concepts for the implementation of this framework in a coherent running system;
- the COMBINE project produced a great number of developer support tools, but limited the undertaken modelling efforts to its specific project window including a few selected tools for energy saving design, without much consideration of the scalability and more general applicability of the approach;
- the EPISTLE project worked out a comprehensive modelling framework including a general ontology, a meta model and a central core model, but envisaging a large shared database and integration techniques that are applicable mostly for global organisations, typical for the petrotechnical industry sector;
- the REMAP project provided a well thought-out revision management system enabling distributed team work, but associates this system tightly with a dedicated object modelling approach, maintaining only a weak link to STEP data models, and ignoring other requirements except version/revision control;
- the RISESTEP project led by Aerospatiale and involving a large consortium of key European industry players elaborated advanced techniques to integrate the disparate PDM, CAD and other IT systems used by its partners in distributed environments, but, similar to EPISTLE, focused basically on solution methods that are appropriate for the global players in the aerospace and the automotive industries;
- the SEED project proposed a novel modular representation approach, but only for a narrow domain (the early design phases), and not specifically related to concurrent engineering;
- the VEGA project focused basically on technological aspects, such as the use of distributed CORBA-based object processing with STEP models, etc.

Thus, whilst there is much prior research related, directly or indirectly, to the subject of this thesis, little work has been done to investigate modelling and interoperability issues in the context of an overarching concurrent engineering framework for AEC.

One of the few major efforts in that respect is the US DICE project (Distributed and Integrated Environment for Computer-Aided Engineering). It examines the use of advanced technologies for cooperative product development, including knowledge-based methods, and several related concurrent engineering issues, such as:

- effective coordination and communication;
- capturing of design rationale;
- forecasting the impact of design decisions;
- collaborative negotiation methods for conflict management.

In addition, greatest attention has been paid in DICE to the flexibility and extensibility of the approach, by assembling the overall capabilities of the system from individual components and capabilities. However, the DICE project ignores current standardisation efforts like STEP and IFC, favouring the development of concepts and methods in an unconstrained prototype modelling environment of its own. It is therefore not likely to be taken up in future building IT developments.

Two other projects that deserve attention are the EU projects COMBI (1993-1995) and ToCEE (1996-1999). These projects exerted considerable influence on the work presented in this thesis due to the active participation of the author as one of the principal investigators in both of them.

The COMBI project (Computer-Integrated Object-Oriented Product Model for the Building Industry) was smaller in scope and number of participating organisations than most of the other projects mentioned above. Its main objectives were initially set up as follows:

- develop a flexible modelling framework capable of representing multiple design perspectives;
- integrate knowledge-based design assistance tools, along with existing numerical and CAD applications;
- contribute to the solution of consistency problems related to evolving design information by capturing the effects of individual design actions.

COMBI focused primarily on integration issues and the examination of selected knowledge-based design methods, but it addressed also several aspects relevant to concurrent engineering in a STEP-conformant way. Such aspects included the transformations between different representations of the same physical objects, dynamic object evolution and re-classification, Internet-enabled data exchange.

The modelling framework of COMBI is less detailed compared to other projects, as e.g. COMBINE or CIMSTEEL, but it takes a broader look at the problem domain and, like ATLAS, incorporates several features that contributed to the creation of the IFC models.

Some of the concepts developed in COMBI were realised by ad hoc written support tools, lacking a formal specification basis, which limited their broader applicability. However, although the COMBI consortium was dissolved after the end of the project, many of the suggested ideas have been further developed in the ToCEE project and in this research work.

ToCEE (Towards a Concurrent Engineering Environment in the Building and Civil Engineering Industries) had a much wider scope compared to its precursor, COMBI, and is one of the few projects that examined the principal construction of an IT framework for concurrent engineering as a primary project goal. It addressed key issues for a successful concurrent engineering approach, including:

- a unified modelling framework for distributed process, product, document and regulation data;
- legal aspects related to the project data and the electronic documentation;
- information logistics and communication management;
- inter-discipline conflict management;
- monitoring and forecasting;
- elements of cost control.

Several domain problems that were previously examined in isolation were considered in ToCEE together, to provide a seamless multi-client/multi-server prototype environment where a wide range of tasks were inter-related. The layered modelling framework of COMBI was enhanced with two additional layers and encompassed a broad spectrum of data, including information about the information system itself (Turk et al. 2000). However, many aspects related to model-based product development in the line of concurrent engineering methodology remained without due consideration in the ToCEE project as well, and there was little work done for the systematic modelling of interoperability aspects w.r.t. the overall system. The exploration of such issues forms a major part of the work presented herein.

1.4 Demarcation

The creation of an IT environment providing comprehensive concurrent engineering support can be expected to include a large number of specialised components of great complexity. The development of all these components by one person is certainly not manageable in a lifetime. However, as already mentioned, many aspects of the envisaged environment have been separately investigated in previous research efforts, a number of related methods are known from computer science, and a variety of support tools are available from the software industry. All this does not have to be re-invented. Therefore, the *emphasis of this thesis* is on issues that are not yet sufficiently clarified, and are only partially addressed in existing approaches. This includes: (1) the principal structure of the modelling framework of the environment, (2) the specification of a common communication paradigm that can easily be applied by heterogeneous software components, (3) the basic interoperability of these components on systemic and semantic level, and (4) the choice of an appropriate server-side representation of the project model data, that matches the requirements of the environment.

Out of scope is the development of product data models and dedicated engineering applications. Out of scope are also data management issues requiring deep domain knowledge. Such aspects should be treated on the level of application systems, or by sophisticated server agents, extending the basic server functionality addressed in this study. Interoperability on functional level, i.e. the specific reaction of the concurrent engineering system to user or application actions has been studied in principle, and sample tools have been developed that are demonstrated in the implemented system prototype. However, the programmed algorithms need further investigation and are only briefly introduced in the presentation.

Work on some of the abovementioned issues began in the COMBI and the ToCEE projects as a cooperative effort of several persons including the author. This work is related to the implemented modelling framework outlined in sections 3.3 and 3.4 (developed with contributions of E. Ammermann, R. Junge, T. Liebich and R. J. Scherer from COMBI, and R. Amor, J. Hyvärinen, R. J. Scherer, Ž. Turk and R. Wasserfuhr from ToCEE), as well as to the Information Container and the Communication Model specifications provided in sections 4.2 and 4.3 (developed with the contribution of R. Wasserfuhr). However, due to some differences in individual ideas, the concepts discussed in these sections diverge, at some places considerably, from the respective concepts described in the reports to these projects.

1.5 Approach

As a whole, the research presented in this thesis can be seen as a **problem-solving task**: it tries to develop a coherent solution to a set of problems related to the operability of the distributed, highly fragmented world of CAE in building construction, by providing concepts, specifications and methods for a model-based concurrent engineering IT environment. The core of the developed approach is a suggested novel method for model mapping based upon a formal, declarative mapping language combined with object-oriented and knowledge-based techniques.

However, due to the broadness of the target domain, the variety of involved IT aspects, and the vague definition of user requirements from the practice, it was not possible to find a closed-form solution derived from a formal axiomatic theory. Therefore, a research method based on the **theory of empirical science** has been selected. In spite of many iterations, try-and-error steps and reformulations that accompanied the research work, it can be generally delineated by the eight phases shown on fig. 1.3 below.

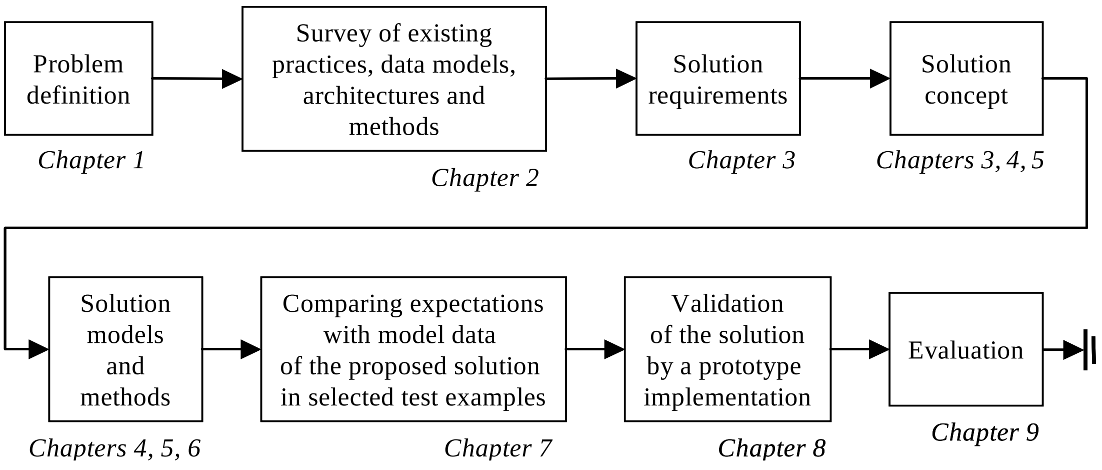

Problem definition
Chapter 1
Survey of existing practices, data models, architectures and methods
Chapter 2
Solution requirements
Chapter 3
Solution concept
Chapters 3, 4, 5
Solution models and methods
Chapters 4, 5, 6
Comparing expectations with model data of the proposed solution in selected test examples
Chapter 7
Validation of the solution by a prototype implementation
Chapter 8
Evaluation
Chapter 9

Fig. 1.3: Principal phases of the adopted research approach

The discussion of the developed concepts and methods in the following chapters is aligned as much as possible with these phases. Hence, fig. 1.3 can also serve as a quick-reference guide through the thesis.

1.5.1 Basic definitions

For a new discipline, with a history of a little more than 50 years, it is not surprising that the terminology used in the area of information technology is not yet consolidated. Many concepts are often named differently by different researchers, developers and users, and the same terms are often used to name different concepts. In many cases concepts are only vaguely defined, leaving a considerable freedom of interpretation to the reader. In the subdomain of computer-aided building design and construction, as well as in the more specific area of conceptual modelling of buildings, this problem is even stronger because of the inter-discipline character of the research work. Therefore, it is useful to introduce in advance the meaning of some basic terms and notational conventions the way they are used in this thesis.

1. In the context of this study **Concurrent Engineering** (CE) should be understood as the coordinated work of design teams distributed across space, time and organisational boundaries. This coordinated work should be organised in such a way that the main principles of the concurrent engineering methodology, as defined in the relevant literature (see e.g. Winner et al. 1988; Carter & Baker 1992; Kusiak 1993; Prasad 1996), can efficiently be applied. Thus, in this thesis, the emphasis of CE is on the issues related to *computer-supported coordination and communication*. The goal is reduction of total development time by enabling continuous interaction between the disciplines involved in a design project.

2. By a **Concurrent Engineering Environment** (CEE) I shall understand the information technology environment in which CE is practised.

   A **CEE System** is an IT system which enables CE in CAE/CIC.

   Computer-integrated construction (CIC) is both the use of computers for all kinds of applications and the *integration* of these applications via data transfer networks and standards (Howard et al. 1989).

   Seen in this perspective, **integration** means efficient information sharing and data exchange using IT as enabling technology (Björk 1995).

   A prerequisite for the development of information sharing and data exchange standards is the development of formalised *conceptual models*. Each such model represents some real world abstraction, based on an underlying *universe of discourse*.

   In accordance with that, **Universe of Discourse** (UoD) is this delimited part of the real world which encompasses the set of objects representing the knowledge of the domain in the subject area of an information modelling effort. In other words, the UoD is a well-formed abstraction of reality.

   A **conceptual model** is the formal representation of the UoD in terms of a set of strictly defined linguistic symbols and a set of necessary logical propositions on these symbols. Often, instead of conceptual model, *conceptual schema* or simply *schema* is being used. Here, *schema* is also the most frequently used term, mainly because of its shortness.

3. The conceptual model of a particular class of artefacts, in our case a building, a constructed facility, or any self-contained part of these, is known as the **Product Data Model** of that class of artefacts. However, in the literature both product data model and product model are often used interchangeably. This may cause confusion, especially where both terms are distinctly needed. To provide for the necessary distinction, the definitions given in (Björk 1995) have been adopted here as follows:

   - A **building product data model** (shortly: *product data model*) is a type of conceptual schema where the UoD consists of buildings throughout their design, construction, operation and maintenance. A building product data model models the spaces and physical components of a building directly and not indirectly by modelling the information content of traditional documents used for building descriptions.
   - In contrast, a **building product model** (shortly: *product model*) is the information base describing some particular building. The structure of the information base is in conformance with a precisely defined conceptual schema (the corresponding building product data model).

Thus, concisely, a product data model represents the information structure that can be used to model any particular product of the referenced class of products. A product model is the *instantiation* of a product data model.

4. **Product Data Technology** (PDT) is the specific IT area related to the development of product data models. However, research and development in the last years have shown that the conceptual modelling of products and the related production processes are closely associated, and cannot be exercised independently. As a result, a number of suggestions for integrated project (process and product) data models have emerged (cf. Luiten et. al 1993; Fisher & Froese 1996). The current IFC documentation also refers to project, and not product data modelling.

   In fact, the emphasis of conceptual modelling has never been on product data, which is merely the dominating part of the data needed in a project, but on the adequate structuring of the information addressed in the identified UoD. Thus, PDT should be understood as equal to project (product and process) data technology. Similarly, instead of product data management and product data services, we shall speak of project data management and project data services, in both cases with emphasis on the word “data”.

5. Today, conceptual models are generally developed on the basis of the **object-oriented modelling paradigm** which dominates the field of software engineering at least since the proliferation of applications using the C++ programming language.

   This paradigm introduces several terms reflecting its outstanding characteristics, such as *object*, *class*, *instance* and *inheritance*.

   An **object** is a “thing” with individual identity, properties and behaviour that distinguishes it from other objects (Russel & Norvig 1995).

   In the conceptual models developed in ISO STEP and by the IAI the term **entity** is used instead of object with a similar, yet slightly different meaning. In the context of this thesis, *entity* is viewed as fully identical with *object*.

   A **class** (object class, entity class) is a collection of objects with common structure, common behaviour, common relationships and common semantics (Rumbaugh et al. 1998).

   The behaviour of a class is represented by its **operations**.

   The structure of a class is represented by its **attributes**.

   Each class describes a possibly infinite set of individual objects; each such object is said to be an **instance** of its class.

   A class can be defined broadly, and then refined into successive finer **subclasses**.

   Each such subclass **inherits** all the properties (structure and behaviour) of its *superclass* and adds its own unique properties to it. Thus, the instances of a *subclass* are in fact a subset of the instances of its superclass.

6. The set of all defined classes within a single product modelling effort, together with their properties and inter-relationships, comprises the *specification of a product data model.*

   The set of all actual instances of these classes, together with their actual properties and actual relationships, comprises an actual *instantiated product model.*

Ideally, the **Modelling Framework of a Concurrent Engineering Environment** should be the set of all related conceptual models that enable the operability of its components, both individually, and as one consistent whole. Such frameworks do not exist yet. One of the main efforts of this thesis is to suggest how such a framework can be constructed and implemented.

1.5.2 Presentation formalisms

Some of the next chapters contain formal specifications, definitions, functional descriptions and implementation examples presented with the help of recognised graphical and textual notations from the fields of mathematics, logic and computer science. However, in the literature different notations can be found for one and the same concept, and the same notational symbols used for different purposes.

The notational conventions used in this thesis are as follows:

1. **Full and partial model schemas** are presented in EXPRESS/EXPRESS-G in accordance with (ISO 10303-11 1994). Even if at some places UML (Rumbaugh, et al. 1998) or OMT (Rumbaugh et al. 1991) might have been a better choice, EXPRESS/EXPRESS-G has been given preference because: (1) it is the presentation formalism adopted by ISO STEP and IAI/IFC (IAI 1999c), (2) it is widely used in many research reports and publications, and (3) it is a pure conceptual modelling language, whereas OMT and its successor UML are more focused on software development techniques.

2. **Syntax specifications** are shown in EBNF (extended Backus-Naur format).

   BNF-like productions are commonly used for such purposes, but there are many different variations known. In this thesis, a BNF-production is expressed as:

   ```
   term = description .
   ```

   where **`term`** designates the new concept (language element) being defined, "**`=`**" stands for "*defined by*" and "**`.`**" ends the production rule.

   In the **`description`**, words or symbols that stand for themselves and should be written as such are enclosed in apostrophes. In contrast, a word not enclosed in apostrophes denotes a term defined elsewhere in the syntax, and should be thought as substituted by its respective description at all places where it appears on the right hand side of a rule.

   Except where enclosed in apostrophes, *brackets*, *braces*, *asterisks*, *plus signs* and *vertical bars* are metasyntactic marks. Parentheses are *not* used as metasyntactic marks because they appear very frequently as language elements themselves.

   Brackets, **`[ ]`**, indicate that what they enclose is optional. *Braces*, **`{ }`**, simply parenthesize what they enclose, but when followed by an asterisk, **`*`** , denote that the enclosed element may appear 0 or more times, and when followed by a plus sign, **`+`** , one or more times respectively. A *vertical bar*, **`|`** , stands for 'OR'. However, within braces or brackets it serves merely to separate mutually exclusive choices.

   In summary, `{x}*` means 0 or more occurrences of `x`, `{x}`$^+$ means 1 or more occurrences of `x`, and `[ x | y ]` means 0 or 1 occurrences of either `x` or `y`. For example, in the rule:

   ```
   schema_mapping_def = '(' MAP [ PARTIALLY ] { schema }+
                             FROM { schema }+ mapping_spec_set ')'.
   ```

   `MAP`, `PARTIALLY`, `FROM`, `schema` and `mapping_spec_set` are supposed to be defined in other production rules, whereas the parentheses must be written as such; `schema` may appear 1 or more times after `FROM`, but `mapping_spec_set` is expected exactly once. Capitalised words like `MAP`, `PARTIALLY`, `FROM` are generally defined as keywords in the form: `MAP = 'MAP'.` Capital letters are used to emphasise that fact.

3. **Logical and relational expressions** are presented by standard arithmetic and first-order logic symbols, extended with some well-known mathematical notations for sets, lists, pointers and arithmetic to make them more compact. Such extensions, also called *syntactic sugar*, are often found in the literature in different variations. With minor adaptations, the notations in this thesis are taken from (Russel & Norvig 1995) as follows:

Syntactic element	Notation	Examples
Propositional logic:		
Negation (not)	$\neg$	if `P` is true, then $\neg$`P` is false
Conjunction (and)	$\wedge$	`P` $\wedge$ `Q` is true when both `P` and `Q` are true
Disjunction (or)	$\vee$	`P` $\vee$ `Q` is true when either `P` or `Q` is true
Implication (if-then)	$\Rightarrow$	$(P \vee Q) \wedge \neg Q \Rightarrow P$ (modus ponens)
Equivalence (iff)	$\Leftrightarrow$	$\neg(P \vee Q) \Leftrightarrow \neg P \wedge \neg Q$
First-order logic:		
Universal quantifier (for all)	$\forall$	$\forall$ x `Door(x)` $\Rightarrow$ `BuildingElement(x)`
Existential quantifier (there exists at least one)	$\exists$	$\exists$ x `belongs(x,Site)` $\wedge$ `Building(x)`
Unique quantifier (there exists exactly one)	$\exists!$	$\exists!$ x `Identifier(x)`
Equality	=	`Building(x) = Building_1`
Sets and lists:		
Empty set	$\varnothing$	
Adjoin (x, EmptySet)	**{ x }**	{ `Building_1` }
Adjoin (x, Adjoin(y,EmptySet))	**{ x y }**	{ `Building_1 Building_2` }
Union (R, S)	$R \cup S$	{ `c1 c2` } $\cup$ { `c3` } $\Rightarrow$ { `c1 c2 c3` }
Intersection (R, S)	$R \cap S$	{ `c1 c2` } $\cap$ { `c1 c3` } $\Rightarrow$ { `c1` }
Member (x, S)	$x \in S$	`c1` $\in$ `Column` or `c1` $\in$ { `Column` }
Subset (R, S)	$R \subset S$	`Column` $\subset$ `BuildingElement` ; { `c1 c2` } $\subset$ { `Column` }
Object-Oriented Modelling:		
Class X	**X**	`IfcProduct`
Class X in schema S	**S:X**	`IfcKernel:IfcProduct`
Instance x_i of class X	$x_i \in X$	`W1` $\in$ `IfcWall`
All instances of class X	**{ X }**	{ `IfcWall` }, or just `IfcWall`, when unambiguous in the context
Attribute a of class X	**X.a**	`IfcMaterial.Properties`
Attribute value a of instance x_i	x_i**.a**	`IfcSystem1.GroupPurpose = "Lateral Structural System"`
Pointer (ref. link) to x_i	$\uparrow x_i$	$\uparrow$`IfcSystem_1`
Mapping transformations	$\rightarrow$	Class mapping: $X \rightarrow Y$ Attribute mapping: $X.a \rightarrow Y.b$
Functional transformation	$\underset{F}{\rightarrow}$	$b,d \underset{F}{\rightarrow} a$, where: $F(b,d) = b * d$

1.6 Demonstration Examples

In order to demonstrate and validate the proposed approach, appropriate demonstration examples had to be selected. It was not possible to invent a single, consistent example that could be tracked through all parts of the thesis because of:

1) the broad scope of the targeted problem domain,
2) the focus of the work mainly on server-side methods and tools, which can only be demonstrated with the help of adequate external client applications[*)],
3) the need for dedicated, artificially constructed test cases to expose specific system features, which would otherwise require lengthy, sophisticated, and difficult to understand usage scenarios, and
4) the need for comprehensive graphical output generated by dedicated third-party applications to illustrate, in concise form, the result of certain server operations.

Therefore, different use cases - adapted from the work of the author in the EU projects COMBI and ToCEE, taken from the IFC documentation (IAI 1999b, c), or from other known publications, have been used as appropriate, along with several short examples, specifically designed to facilitate the discussion of certain details of the suggested approach.

1.6.1 Examples adapted from the COMBI project

Based on a case study performed in the COMBI project, these examples primarily focus on methods and techniques for *cooperative work and design tool integration*.

The original COMBI case study, shown at the final workshop of the project, presents a simplified version of the Olympic Airways Flight Simulator Building at Thessaloniki, Greece (fig. 1.4). The examined design process includes the definition of the initial architectural design data, and the interaction between structural preliminary design, structural analysis and foundation design. It is described in detail in (Scherer & Sparacello 1996).

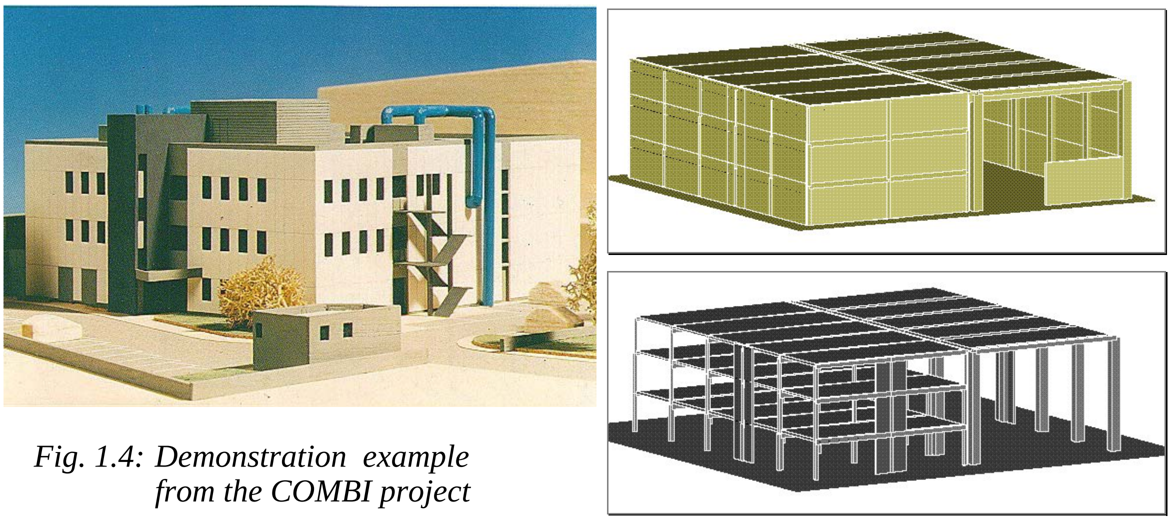


Fig. 1.4: Demonstration example from the COMBI project

(*Left*: model of the finished building; *top right*: screenshot of the primary architectural elements; *bottom right*: screenshot of the bearing system of the building)

*) As already mentioned, the development of client applications has not been in the focus of attention of this study.

The conceptual product data models used in COMBI were defined from scratch (Ammermann et al. 1994; Katranuschkov 1995). They were not further developed after the end of the project, but the ideas of the overall COMBI framework have significantly influenced the later IFC development (see Junge & Liebich 1998; IAI 1999b).*)

1.6.2 Examples adapted from the ToCEE project

These examples, drawn from the ToCEE final workshop demonstration, focus on the concurrent engineering approach concerning *the communication and collaboration of distributed design teams* in a building construction project. The case study presents a simplified version of Hall 21 of the New Munich Fair facilities in Germany (fig. 1.5), involving the domains of architecture, structural, HVAC and geotechnical engineering, as well as facilities management. Unlike the example from COMBI, the emphasis in this case study has not been on the coordinated use of application software, but on the collaboration between the different professionals in a distributed project team. Therefore, the developed scenario, trying to provide an adequate demonstration of the environment and the component applications (mostly on CAD basis), primarily examines: (1) issues related to parallel, concurrent working, (2) creating, managing, coordinating changes, (3) information management in a client-server system with both locally and globally maintained model data, and (4) selected project management issues.

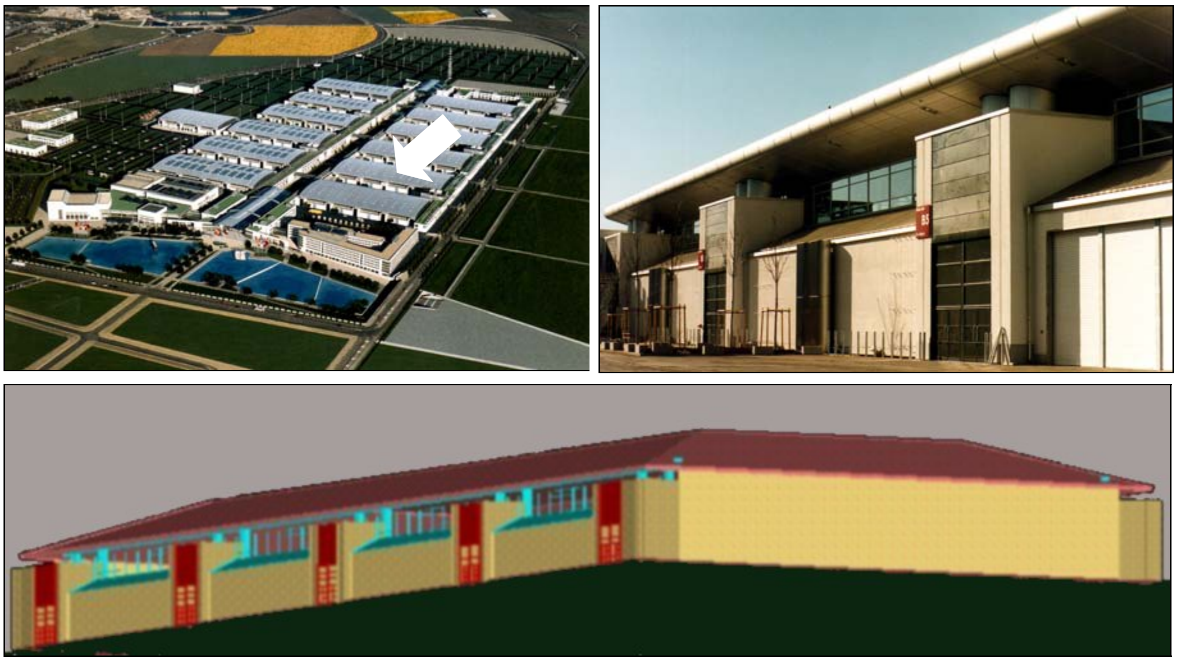


Fig. 1.5: Demonstration example from the ToCEE project

(*Top left*: bird-eye view of the New Munich Fair facilities, showing the location of Hall 21; *top right*: side view of the building; *bottom*: CAD visualisation of the model data retrieved from the project data server)

*) Appendix VII at the end of this thesis provides a schema level diagram of the COMBI framework, along with a summary of the number and types of modelling objects used.

The original ToCEE scenario is described in detail in (Turk & Scherer 2000). In chapter 8 an adapted version of this scenario is presented, concentrating on the particular methods for project data modelling and management developed in the frames of this thesis. Specific aspects of the studies performed on the basis of the ToCEE scenario are directly or indirectly addressed at several other places in chapter 4 and in appendices II and VII, to back respectively discussed concepts.

The modelling framework of ToCEE uses as baseline the IFC Project Model, Release 1.5 final (IAI 1997). However, in order to provide adequate support to the identified concurrent engineering requirements, some specific extensions to the IFC project model were developed as well, including: (1) a unique object identification approach, (2) enhanced grouping mechanisms, (3) domain-specific views, (4) approval and access right support, (5) system related meta-model specifications (Hyvärinen, Katranuschkov et al. 1999).*)

1.6.3 Examples from IFC 2.0

These examples focus on *the possible use of IFC as an underlying model of the proposed concurrent engineering environment*.

As an emerging industry standard, the IFC project model was given highest priority in the validation of the proposed approach, and it is applied as much as possible as a reference model for the demonstration of the developed data management concepts.

Whilst the ToCEE environment is also based on IFC, the given ToCEE examples are less explicit w.r.t. the specific features of the IFC architecture. Therefore, certain examples had to be (re-)constructed so that the “current” IFC models could be more clearly referenced.**)

In addition, *selected fragments of a study, demonstrating a structural engineering domain model developed as an extension to the IFC Project Model Release 2.0*, are used as well. This study was conducted in the frames of a diploma work under the guidance of the author (Weise 1999). Its emphasis was on the interoperability between core IFC model data and the domain-dependent data needed in architectural and structural design. In particular, it examined (1) how the architectural data can be transformed to provide useful input to structural design, eliminating as much as possible waste times, (2) what extensions to the IFC model schemas are needed to capture at least basic structural design decisions, and (3) what happens when the two domain models (architectural and structural) are used in parallel in the same project.

Detailed specifications of the IFC Project Model and the proposed Structural Domain Extension Model are provided in (IAI 1999b, c) and (Weise 1999) respectively.#)

*) An overview of the ToCEE models is provided also in Appendix VII at the end of the thesis.

**) The word “current” is put in quotations here because of the fast development pace of IFC. Thus, while this thesis was still in its editing phase, the “current” IFC project model evolved to Release 2.x, with 3.0 scheduled for August, 2000. Nevertheless, since Release 2.0 is well documented, and the details of the model are not so essential w.r.t. the concepts developed in this thesis, I hope that the examples drawn from IFC 2.0 can still serve as adequate reference to the general IFC modelling architecture.

#) The EXPRESS-G diagrams of the IFC Kernel Model schema and the Structural Domain Extension Model schema are provided also in Appendix VII at the end of the thesis.

The test model, shown on fig. 1.6, was crafted by hand. The application tools, used along with the project data management services developed in this thesis, were: (1) a general-purpose CAD system (AutoCAD R14) simulating architectural and structural design work, (2) a structural analysis application (SOFiSTiK), and (3) a dedicated client enabling the communication between the project data server and the user applications.

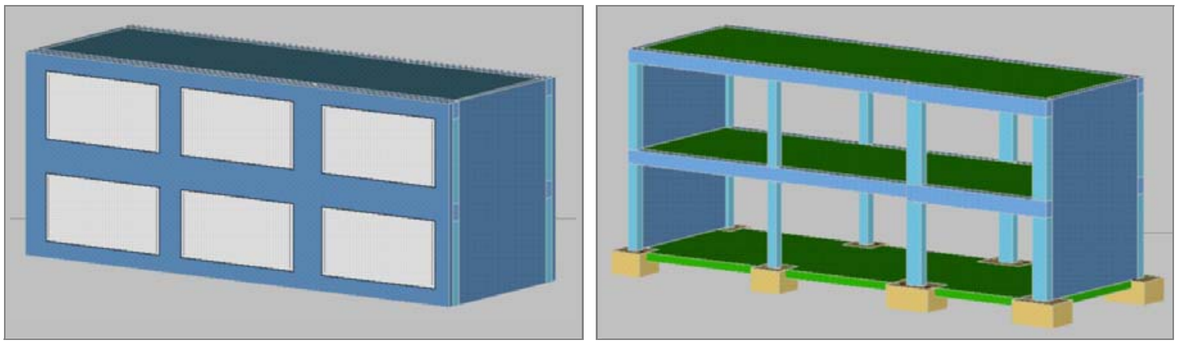


Fig. 1.6: Demonstration example used for the validation of a proposed structural domain extension model for IFC v. 2.0

(*Left*: initial architectural design data of the sample structure; *right*: visualisation of the bearing structure developed in accordance with the proposed structural domain extension model, as retrieved from the project data server)

1.6.4 Other examples

In addition to the above, several examples have been specifically designed or adopted from other sources to illustrate individual concepts, and a case study conducted by (Liebich et al. 1995) to demonstrate the capabilities of existing model transformation approaches has been used in the validation of CSML, the context-independent schema mapping language developed as part of this thesis.

For quick reference and preview, all given examples are summarised below:

Chapter # / **Examples**	**4**	**5**	**6**	**7**	**8**	**App.**
Adapted from COMBI	1	–	–	①	1	-
Adapted from ToCEE	2	–	–	-	① + 5	3
Adapted from IFC 2.0	3	6	1	①	-	8
Others	3	3	6	①	-	6

Notes:

A number in circle denotes a more comprehensive case study; plain numbers refer to short specifications or (pseudo) code sequences, and/or selected illustrative screenshots.
The short examples given in chapter 5 are all included in tables 5.3-5.7.

Chapter 2: Background

Memory: what wonders it performs in preserving and storing up things gone by, or rather, things that are!
– Plutarch, Morals: On the Cessation of Oracles

In the conceptual development of a complex environment many important design decisions have to be taken at the very outset. To justify such decisions, and to help the reader understand them better, it is first necessary to examine their background.

The purpose of this chapter is to shed light on the recent research and achievements in the areas that are closely related to the proposed new model mapping approach for concurrent engineering processes.

First, the basic principles of the concurrent engineering methodology and its applicability in the AEC domain are discussed from the viewpoint of building construction practice, and their main implications to building IT are identified.

Second, the state of the art of available software solutions that can be applied in the envisioned concurrent engineering environment is briefly reviewed.

Third, the achievements, the existing different concepts and the current state of research in engineering data modelling are analysed, and the principal perspectives for the use of PDT as enabling information infrastructure for concurrent engineering are outlined.

2.1 Concurrent Engineering from the Viewpoint of Building Construction

The term *Concurrent Engineering* has been introduced for the first time in a report of the US Institute for Defence Analysis (Winner et al. 1988) to express the method of concurrently designing both the product and its production and support processes. That report defines concurrent engineering as *„a systematic approach to the integrated, concurrent design of products and their related processes, including manufacturing and support. This approach is intended to cause the developers, from the outset, to consider all elements of the product life cycle from concept through disposal, including quality, cost, schedule and user requirements“.*

However, while this is still the most widely accepted definition of the goals of concurrent engineering to date, it does not provide many clues about the methodology itself, neither about its more detailed objectives.

In (Dean & Unal 1992) concurrent engineering is identified basically as *„getting the right people together at the right time to identify and resolve design problems”*. Putting clear emphasis on the design phase, the authors conclude that concurrent engineering means in fact designing for assembly, availability, cost, customer satisfaction, maintainability, manageability, manufacturability, operability, performance, quality, risk, safety, schedule, social acceptability, and all other attributes of the product.

This view on concurrent engineering is supported by many researchers. In (Jagannathan et al. 1991) concurrent engineering is defined as the process of forming and supporting multi-functional teams that set product and process parameters early in the design phase. A similar understanding of the focus of concurrent engineering can be found in (Kusiak 1993) and, indirectly, in (Bralla 1996) and (Womack et al. 1991), who provide many excellent examples of the influence of design to the final quality and cost of a product.

As a theoretically founded methodology for product development concurrent engineering has first been applied in manufacturing in the late 1980s, although some authors argue that concurrent engineering practices have been common to manufacturing even before World War II (cf. Ziemke & Spann 1993). Indeed, in the early days of manufacturing new products and their production processes have often been designed simultaneously. This intuitive "concurrent engineering" approach has been a natural way of working for the small and spatially compact teams of that time, but it is hardly applicable to the complex, highly distributed production processes of the present time.

As understood today, concurrent engineering has clear definitions of its context and its essential features, as well as guidelines for its implementation, at least for the manufacturing industry sectors. But how do its concepts apply to construction and what implications do they have to building IT?

2.1.1 Fast or concurrent, that is the question

The main factors for competitive advantage in construction are quality and cost. A project is successful if it is the right thing done right (quality), on time, and within budget (cost).

In **traditional construction** cost competition has meant minimisation of the costs in each phase of a basically sequential process. Managing of quality has been understood merely as supplying of materials and building components in conformity with the design.

Today, the need for rapid completion of buildings that meet the life cycle requirements of the owners has become a new competitive factor in construction. This has led to the introduction of an intuitive method for reducing construction project time, known as **fast tracking**.

The essence of fast tracking is in overlapping of design and construction activities. In many cases, especially by „simpler" projects, it has helped to achieve shorter project duration and decrease the construction costs. However, by planning neglect, i.e. too low percentage of design completion prior to the start of construction, fast tracking can also result in less than optimal design, unexpected cost overruns, and considerable project delays (Laufer & Cohenca 1990; Tighe 1991). In extreme cases, this may even lead to disastrous situations[*)]. Thus, with fast tracking, an undesirable project outcome is not seldom. This is mainly due

[*)] A typical example of the high risk associated with fast tracking is related by (Quaife 1991) as follows: *"The 1988 fast-track Station Square project in Vancouver, Canada, experienced a roof failure on the opening day, resulting from an accumulation of design and management errors. Two years later, an Inquiry Report of the Canadian Project Management Institute attributed the main reason for the failure to the fast-tracking with multiple contracts, describing it as 'the spreading of work and responsibilities among numerous parties in a situation where both owners and prospective tenants are thinking out loud about possible design changes or proposing them directly to engineers, architects ... while the work is proceeding'. The report allows to infer that design and contractual concerns were rationalised as not significant, due to the pressure to complete on time."*

to the insufficient consideration of the relationships between the value and cost of construction and the ability to influence them with project advance, as shown schematically on fig. 2.1 below. The exaggeration of the importance of time is the major drawback of the fast tracking approach, leading to greater *uncertainty* of a project's outcome in comparison to other production methods.

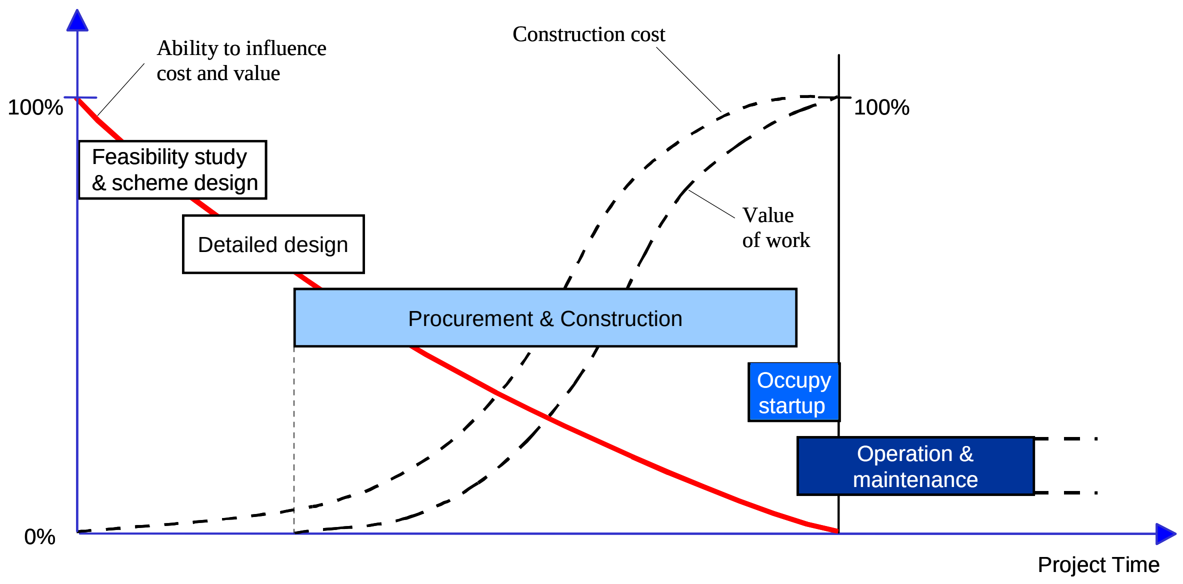


Fig. 2.1: Schematic presentation of the typical overlapping of project phases and their influence on the value and cost of construction / adapted from (Hendrickson & Au 1989) /

Concurrent Engineering, on the other side, is a theoretically founded production methodology. Its major objectives are to decrease the duration of an engineering project, to increase the value of the product, and at the same time reduce its cost. These, partially contradictory objectives are pursued simultaneously and systematically, with constant concern for *uncertainty reduction*.

Concurrent engineering is methodologically much better developed than the fast tracking approach. Hence, it is definitely worthwhile to transfer its methods and techniques to building construction, especially with respect to critical for the building industry aspects like customer satisfaction, team approach, improved use of off-the-shelf design solutions and time management.

Today it is recognised that concurrent engineering implies:

– collaborative, parallel product and process design,
– advanced project management principles,
– consideration of the whole product life cycle,
– effective communication and information sharing across disciplines and phases of the product's development, and
– adequate consideration of responsibilities.

Let us now examine these aspects of concurrent engineering in some more detail.

2.1.2 Concurrent engineering in design

A building product is a complex system developed to satisfy a set of purposes. When life is involved, associated considerations, such as safety and risk assessment, facility management, refurbishment and demolition have to be added as part of the design purposes. Hence, design must be performed *holistically*, taking in consideration many multidisciplinary requirements and goals. However, by tradition, each design trade in building construction has developed an own, *reductionistic* approach, concentrating on its narrow discipline-specific goals, and often failing to look at aspects affecting the other professions involved in the process.

Taken alone, the traditional conceptual basis that sees design mainly as a set of *conversion processes*, i.e. processes of defining form to satisfy certain needs and requirements, is quite restrictive. If exaggerated, this view on design can lead to inefficiency, and even provoke errors. In contrast, as indicated by (Huovila et al. 1994), in the concurrent engineering paradigm the design process *must* be considered simultaneously as:

- conversion,
- information flow, and
- value generation.

The view of **design as a conversion process** focuses on what people do in design. In this perspective, improvement of design means making design activities more effective and efficient. Such objectives have encouraged the use in everyday practice of advanced design software, such as CAD, calculation and analysis programs, knowledge-based decision support tools, and, to some extent, advanced project management methods and tools applied to the design process.

Conversion is the best studied aspect of design, and its most important and natural feature. However, if taken as a sole view of the design process, it may cause serious oversights with respect to those activities that do not directly contribute to conversion, such as inspecting, storing, retrieving, communicating design information, as well as to the fulfilment of long-term customer needs.

The view of **design as information flow** originates from industrial engineering. It focuses on what happens to information in design, by considering four basic types of information flow - *conversion*, *moving*, *waiting* and *inspection.* Of these, only conversion is design proper, while the others are basically not needed and are therefore commonly categorised as *waste*. Thus, in this perspective, improvement of design means to minimise waste, i.e. to reduce the portion of non value adding activities, such as rework due to errors or lack of information, transfer of information, waiting for information, inspection etc. Detailed analyses of the information flows in design have shown that the share of conversion in the total flow time is very little, with typically ½ to 2 ½ rework cycles and more than 30% of the total time spent solely for co-ordination (Dean & Unal 1992; Cooper 1993; IAI 1999a). According to (Northey & Southway 1993), both cost reduction and quality improvement are related directly to cycle time as a primary organisational metric, their basic hypothesis being that „*as many as 90% of the existing activities are nonessential and can be eliminated*“. In this aspect there is much yet to be done.

The major cause for *rework* is uncertainty. Reduction of uncertainty, especially in the early design phases, can be facilitated by: (1) the implementation of appropriate AI-based decision support and simulation tools, allowing to consider interdisciplinary and life cycle aspects beyond the main focus of attention of each individual designer, (2) adequate information exchange paradigms, allowing to capture and communicate design intent, and (3) intentionally delayed design decisions based on a thorough understanding of the overall production process (Ward et al. 1995).

The main problem related to the *transfer of information* is on one hand the information loss taking place during information exchange, and on the other hand the redundancy and/or non consistency of the data. Information loss is most often caused by non compatibility of the design tools and the related underlying conceptual models that are being used. It can be reduced by the application of standardised information exchange formats, methods and tools, e.g. DXF, at the low end, or STEP exchange files (ISO 10303-21 1994) and SDAI (ISO 10303-22 1998) at the high end, but real progress can only be achieved through the development of interoperable environments based on comprehensive interoperability models, methods and techniques. Redundancies and inconsistencies usually arise because information is transferred in large batches between designers or organisations. Such situations can be avoided with the help of information sharing systems on PDT basis, but where organisational borders are concerned, legal responsibilities and contractual issues have to be taken into account as well, and these are not sufficiently supported by today's IT tools.

Long *waiting times* are partially caused by the problems of information transfer, and partially by ineffective planning of activities and ineffective information access. Hence, the reduction of waiting times can be strongly facilitated through the use of advanced communication management and information logistics tools.

At last, *inspection* requires appropriate browsers for filtering out nonessential data, easy navigation and access to important design parameters, and powerful presentation facilities. Such features are already available in advanced CAD systems.

The view of **design as value generation** originates from quality management. Value is generated through fulfilment of the customers needs and requirements. It consists basically of the following three components: *product performance*, *absence of defects* and *ease of maintenance*. Value has to be evaluated from the viewpoint both of the next customer and possible future customers and therefore requires intensive life cycle and facility management considerations. In this perspective, improvement of design means to reduce value loss. The latter may occur due to one or more of the following reasons:

- part of the customer's requirements are missed at the outset,
- part of the design intentions are not properly communicated, and are neglected in later development phases [*)],
- constraints or opportunities of subsequent phases are not taken into account,
- quality control is insufficient, or does not cover all necessary aspects.

[*)] The failure of two skywalks in the lobby of the Hyatt Regency Hotel in Kansas City on July 17, 1981, is cited as one of the greatest structural failures ever to take place in the USA. It was attributed to a combination of three events. First, in progressing from preliminary to detailed design, where joint and connection detailing occurs, the design of the hangers to the spandrel beam connection was inadequate. Secondly, in the assembly drawings, the connection detail was changed by the steel fabricator, sharpening an already critical condition. Thirdly, this second error was not caught during approval checking of the assembly drawings by the controlling structural engineers. These were all errors of communication and coordination in the design process, and focus on documenting the product of design while neglecting „requirements", „process" and „intent" documentation (Sriram 1991).

The reduction of value loss requires the application of IT tools for requirements and regulations management, as well as advanced (AI-based) tools that can be implemented for the support of such aspects as design for manufacturability, design for assembly etc., as well as for the management of conflicts.

However, it is of primary importance not to try to improve the above aspects separately, but to consider them simultaneously, as part of a coherent, conceptually solid environment.

2.1.3 Concurrent engineering in construction

On the one hand, it is obvious that in the procurement and construction phases of a project basically the same principles as discussed for the design phase should apply. However, as indicated on fig. 2.1, even though procurement and construction are of considerably longer duration, they offer by far less possibilities to handle unexpected situations or late discovered errors.

On the other hand, the construction phase is quite interesting to look at, because many of the problems that inhibit the successful deployment of concurrent engineering methodology in the building industry have their roots in the way the *transition* from design to construction is being organised and practised.

In today’s popular **turn-key projects**, dominating the market in most countries, as e.g. in US or UK practice, architects and engineers are hired by the contractor to do certain specific tasks, typically in the conceptual design phase. In this case the contractor is responsible for the whole project management, and he has definite ideas on how to detail the building to make it easy to construct. But he does need someone to perform the detailed design as well, and if he does not have in-house capacities, he will hire architects and engineers to do the job. However, the latter will be operating merely as technicians, and their main responsibility will be to ensure that code compliance is observed. Thus, quite often, customer requirements, life cycle aspects and various project management issues that are related to design remain without adequate concern.

In a **conventional project** architects and engineers are hired directly by the owner, which is a common practice e.g. in Germany. In this case as well, the construction manager will be the one to bid out the job, and, although he will need a lot of input on how to do things, he will with his expertise know how to accomplish many of the tasks, without the architects and engineers having to show a lot of detail. As a consequence, after design development the responsibility is handed over from the architect to the contractor, and the contractor takes command of the decision making processes from then on. This results often in incomplete drawings, change of scope during the construction process, contractual problems etc. A subcontractor who is pricing the job, if confronted with such incomplete drawings, must allow for possible extra costs for work that is not explicitly visible. Thus, his estimate must be “fat” enough to cover all probable eventualities. Or the contractor, if given options as to how to do the job, will choose the one which is most convenient to him, and not necessarily the best for the long-term service of the facility. However, this kind of “value engineering” will do more to give bigger profits to the construction company than value to the customer (Sherwin 1996). As a result, though for slightly different reasons, similar problems as with turn-key projects arise in conventional projects as well, leading, at the end, both to loss of time for the project and to loss of value for the product.

2.1.4 Relationship of concurrent engineering to project organisation and management

Good project management is a major prerequisite for the success of a construction project. It is responsible for „filling the gap“ between design and construction, and is in fact more closely related to design than traditionally anticipated.

A wide-spread (and wrong) view on project management is that it is not more than scheduling and subtrade coordination which takes place during project realisation, i.e. project management starts „when you put the first shovel in the ground“. Indeed, most project problems come to the surface during construction. However, for a good probability of success, an estimate provided by (Quaife 1991) known as the 80:20 rule, says that *„the planning phases expand less than 20% of the total project management effort but must achieve a 'nest-egg' of about 80% of the preparedness for project success. 80% of the project management effort follows during implementation, both to hatch the nest-egg and to realise the remaining 20% of success“*.

The ASCE manual „Quality in the Construction Project“ states that *„more design deficiencies result from mistakes in the management of a project than from purely technical errors“*. Inadequate information, communication or coordinated forethought in the early phases of a project inevitably lead to rework of the design, and to wasted construction activities. The more serious the time pressure, the greater is the risk of such management errors, and the greater the likelihood of „more haste, less speed“.

To avoid such problems, it is necessary to recognise and assess properly, along the line of concurrent engineering methodology, the influence of the separate **elements of integrated project management** on the success of a construction project.

Scope is the element most exposed to planning neglect. The process of fixing the overall project requirements must both lead and interact with the product's design. More detailed requirements must be defined ahead of tendering dates, and all details must be finalised for the preparation of working drawings. Scope definition must be systematic and rigorous, must involve all those with significant interest in the outcome and must achieve their timely sign-off. Changes can be reduced, before sign-off, by a review of the consistency of the requirements and the design solutions (design as value), and the constructability and maintainability of the design.

Quality is in a sense part of the project scope. The goal of quality control is “zero defects first time”, avoiding the biggest problem of planned durations - rework - whether it appears as an unanticipated plan revision, or as a corrective action following the discovery of a defect. Thus, project management problems are more likely when the project manager has no responsibility for quality management, or has too much authority and may be tempted to endorse reduction of quality as a sacrifice to save time.

Activity and resource planning and monitoring are needed to measure productivity and control costs. Productivity and its trends provide important information for assessing and predicting progress, and forecasting the final costs. Where project progress is not measured for the activities in progress there is a temptation to assume that the physical percentage of completion equals the percentage of allowed resources spent. Exaggerated reporting of the physical percentage of completion can hide the true cost and duration trends of a project and hence bring it in danger.

Communication is a commonly recognised criterion of integrated project management. However, except personal communication skills, a less recognised, but strongly needed requirement is the use of a *formal communication system*. Communication shortcomings can threaten efficiency, scope optimisation and quality. They can therefore threaten cost, schedule and performance[*)].

Risk is another important element of project management, which is often neglected to suit the management's desire to approve a fixed project cost, especially when there is a preset budget limit. This is less dangerous for routine projects, but even then a review of potential problems may reveal risks that are not routine at all. Management, after approving a project, has to accept whatever surprises may come from the initial uncertainties, whether recognised at approval time or not. Such "surprises" can be considerably reduced by the application of appropriate simulation and conflict recognition IT tools.

Time is commonly anticipated as the most critical factor in a construction project. Time management involves the early recognition and solution of scope and design problems, and the adequate estimation of their implications for a realistic control schedule. „Communication" and „monitoring" are crucial in scheduling, and manually prepared schedules may respond too slowly to project needs. However, an organisation that uses only this last of all mentioned project management elements, believing that „scheduling by computer" will do the job, will normally experience no better results than with simpler ad hoc methods. Certainly, time is the most elusive of all project management goals because no matter how or when a problem arises it takes time to solve, and there is a limit to how much lost time can be recovered. But if delays that originate from scope, quality, uncertainty or communication problems are treated only as scheduling issues, there is only symptom control, but no real prevention and cure. Therefore, project success is probable only when problems in all these areas are anticipated early in their entirety, and are solved long before becoming critical or, better, by avoiding them at the outset.

2.1.5 Life cycle management

We all know that the development of most technical products begins with requirements and feasibility studies and ends up with mass production and marketing, supported by an appropriate supply chain. Similarly, a construction project begins with the business plan and the brief, proceeds with tendering and construction, accompanied by establishing adequate supply lines, and is finished when the target one-off product is build and occupation can start up (see fig. 2.2). However, the life cycle of any technical product extends far beyond its initial development, and includes all activities to bring forth, sustain and retire the product. For building construction this means that facility management, refurbishment and demolition must also be taken into account. Though such activities lie outside the time span of a construction project, they should not be considered outside its

[*)] The UK Royal Commission to the fatal structural failure of the Melbourne Yarra Bridge in 1970 pointed to a breakdown in communication due to animosities as a major contributing factor. Not only did the responsible site managers allow this to remain unresolved, but their superiors did nothing to solve it - if they were aware of it from their offices in England. This extreme consequence demonstrates the dangers that a project is exposed to if project information is confusing or incomplete, or if normal healthy difference of opinion is allowed to escalate into non-communication (Quaife 1991).

scope. In accordance with the concurrent engineering methodology, the development of a building product, especially in the design phase, must be done in consideration of the long-term aspects of its life cycle. This implies as early as possible use of simulation and facility management tools to forecast future behavioural aspects and take appropriate measures for the regular functioning and maintenance of all component systems.

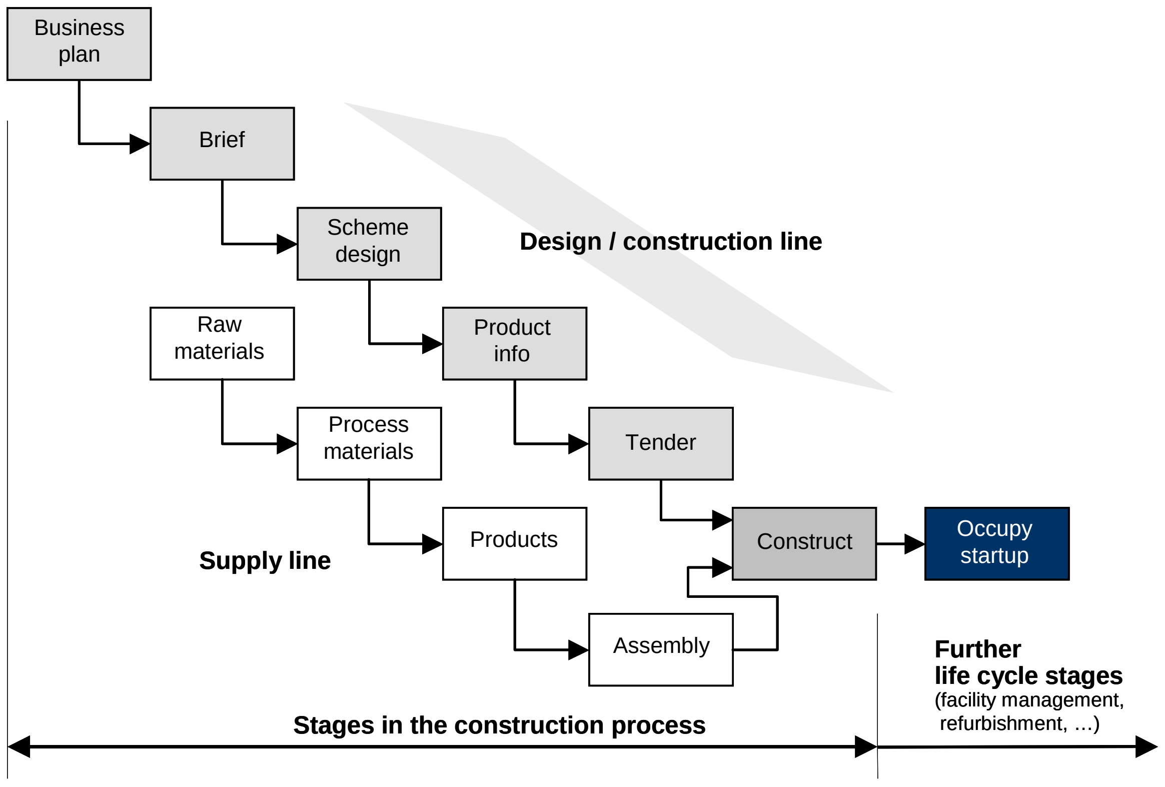


Fig. 2.2: Principal life cycle stages of a building product

Although life cycle management cannot be delegated exclusively to the designers or the contractor, the responsibility for many serviceability aspects is clearly defined in each project, and many quality control issues, expressed through various parameters, are specifically defined in contractual agreements, regulations and codes of practice. Thus, the need for integral treatment of design, construction, project and life cycle management leads to the last but not least aspect of building development - the aspect of responsibilities.

2.1.6 Responsibilities in construction projects

As buildings and construction facilities are becoming more and more complex, no single player can be expected to be capable of executing the design process on his own. Design is handled by a team of separate design professionals with the architect taking charge of the overall view of spatial planning, volumetrics and context, the structural engineer looking after the design of the load-bearing elements, the services engineer handling the mechanical, electrical and hydraulic systems and so on.

During the earliest stages of the design process, the architect may well dictate his requirements for structural grids, column and beam sizes, storey heights etc. to the structural engineer, who will provide him with a schematic response. Once the architect has accepted the principle structural solution, the responsibility for the design returns to the structural engineer, and the architect cannot make any further alterations to the structure without consulting and obtaining permission from the responsible ‘owner’ of the structural design - the structural engineer. Thus, in effect, the architect provides the engineer with performance requirements for the structure which he accepts and responds to. A similar process takes place between the architect and the services engineer, and between the services engineer and the structural engineer. Any change to the performance requirements of the structure, the services, or the architecture needs the sanction of its “owner” in order to ensure that responsibility remains with and is accepted by that “owner”.

Much detailed design is represented by performance based specifications. The designer describes how the building should provide a particular service, such as capacity for the cladding to withstand certain wind loading, the air handling plant to provide a required number of air changes, structural steel junctions to withstand shear etc. These will often form the basis for tender documentation issued to specialist contractors or suppliers who, when appointed or selected, will take on responsibility for ensuring that the product will satisfy the required performance. The contractor may offer alternatives that may vary the specification for acceptance by the original designer, who will probably only do so providing the responsibility remains with the specialist who knows the most about the particular addressed performance characteristic of the product. However, if not strictly controlled, such *delegation of responsibilities* can be dangerous, especially in multi-discipline coordination and the overlapping of design and construction phases, i.e. where uncertainty is high. A typical example for that are the problems which often appear in the coordination of geotechnical, foundation and structural works[*)].

The bottom line of this concise review of concurrent engineering is that, in accordance with the strongly emphasised holistic approach to product development, all the issues discussed above have to be observed simultaneously by the design of an appropriate IT infrastructure to warrant their coherent treatment in the realisation of a run-time CEE system.

Of course, it cannot be required that a CEE system should provide all needed features at once. It is hardly possible to imagine that such a “super system” can be achieved within a single development effort. However, if one or more of the aspects of concurrent engineering are ignored during the conceptual design of the system, there is a high potential danger that these aspects would never be tackled in future system extensions.

Taking into account the enormous development costs, even for a prototype implementation, such planning neglects can hardly be justified.

[*)] The Grand Teton Dam, an earth fill structure more than 300 feet high in Idaho, failed during the initial reservoir filling requiring months of rework. The mode of foundation failure has been extensively analysed, and problems which „may not be directly related to the failure“ but contributed to it have been identified as: dual reporting arrangements, fragmentation of responsibilities including separation of construction from design, and inadequate coordination with field geologists for decisions on the foundation construction.

2.2 IT Support for Concurrent Engineering in Building Construction Practice Today

IT tools that can be used for concurrent engineering processes may take a variety of forms. Schulz (1996) argues that almost any software tool and data format more structured than simple e-mail can qualify and, *if appropriately used*, can be helpful for certain specific needs. Therefore, by the conceptual design of a concurrent engineering IT environment it is useful, at the outset, to take a look at the whole complex of software tools that are available in building construction practice today, and assess their relation to concurrent engineering methodology.

For that purpose, a survey of the state-of-the-art of building IT applications from the viewpoint of concurrent engineering was performed during 1996-1997 on the basis of the following sources:

– published results of comprehensive relevant studies conducted in the last years in several industry countries (USA, UK, Germany, Denmark, Ireland), providing detailed data about the penetration of IT in construction (Häußler-Combe 1994; Latham 1994; CIRIA 1995; Cragg & Zinatelli 1995; Gardner 1995a, b; Healy & Orr 1996; Howard 1996; Schulz 1996; Sørensen & Andersen 1996);
– literature study of magazines, brochures and available on-line material;
– data obtained from partner organisations in the EU projects COMBI and ToCEE[*)].

Details of that survey are provided in (Katranuschkov et al. 1997a). The main issues relevant to the content of this thesis are discussed below.

2.2.1 Key issues

At first, it has to be acknowledged that the building industry has been relatively slow in adopting the new achievements in information technology, the basic reason for that being lack of finance, not lack of interest. The EU project ELSEWISE estimates the annual investments in innovative IT products by the European building industry to as little as 0,4% of turnover (Sauce et al. 1997). Some US sources mention figures of 0,5% to 1%, which is not much higher as well.

Nevertheless, according to the UK report "Building IT 2005" (Howard 1996)[**)] *„it is wrong to think that the construction industry lags behind other industries in its deployment of IT simply because its investment appears to be low. Construction is not a homogeneous industry with steady and regular lines of supply, so comparisons with manufacturing, process and retailing industries are of limited value.“*

[*)] *Building Research Establishment* - for the UK; *Obermeyer Planen + Beraten, Leonhardt, Andrä und Partner* and *SOFiSTiK* - for Germany; *KUPARI Oy* and *VTT* - for Finland; *General Construction Company* and *SOFiSTiK Athens S.A.* - for Greece.

[**)] "Building IT 2005" is a major report from the Construction IT Forum examining the future use of information technology in the construction sector up to the year 2005. The report itself is an innovation in publishing for the construction industry. It collects together the experience of 32 experts in different aspects of construction and/or IT.

“Building IT 2005” raises also several important issues relevant to the current and future use of IT as an enabling infrastructure for concurrent engineering:

1) *Unless there is more investment in IT by construction*, improvements will end up as responses to external imperatives rather than the intrinsic needs of the industry.
2) *Current commercial IT* is sufficiently advanced to meet most needs of the industry.
3) *The Internet*, a cheap, universal communications network waiting to be exploited by the construction industry, is a prime example of already-available technology.
4) *A common language* - The biggest technical issue for construction IT is the way in which information is structured. There is need for a common language across the whole breadth of construction.
5) *Most likely to inhibit systems integration in construction* will be organisational, cultural and industrial factors, and not technological issues.
6) *The technological key to enhanced construction capability, client satisfaction and competitiveness is having systems to gather, organise, transfer and display the right information to the right people at the right time.*

Whilst some of these issues are only indirectly related to the objectives of this thesis, issues #3, #4, and especially issue #6 have been among the main actuators of the research work.

Let us now examine in some more detail the state-of-the-art in building IT in view of the rapid technological progress of the last few years.

2.2.2 The landscape of building IT

Although it was difficult to obtain precise information from surveys mainly based on questionnaires and expert judgement, the analysed data showed good convergence and allowed to deduce more or less stable trends, at least for the next few years.

The investigated issues were selected on the basis of two criteria: (1) reliable amount of raw data, and (2) pertinence to the purposes of this study.

Fig. 2.3 below sheds light on the scope of the examined material, and table 2.1 provides a summary of the collected data by categories. Included in column 2 of this table are the calculated standard deviations which take into account the differences in the results quoted in the investigated publications and the gathered data by the author.

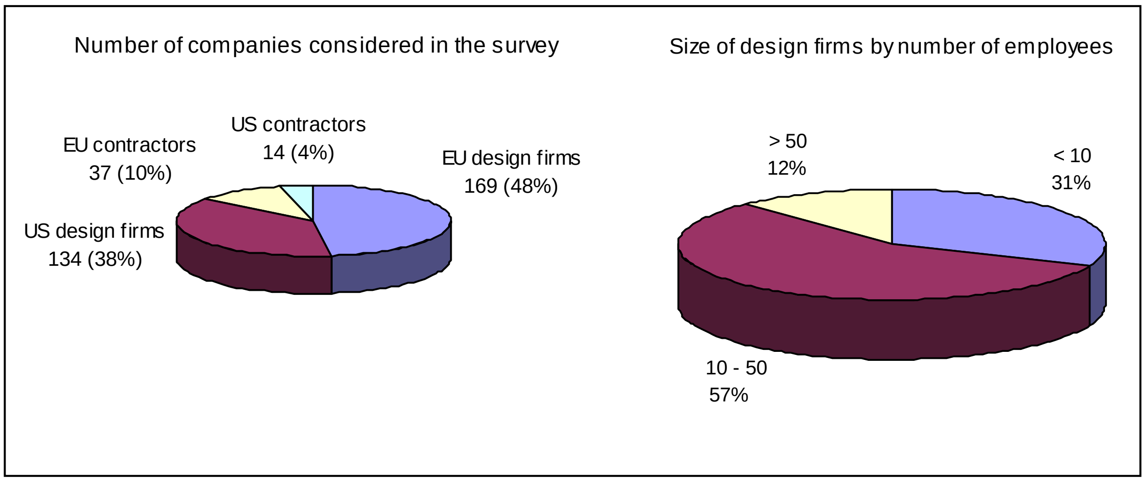


Fig. 2.3: General data of the survey of building IT application

Table 2.1: Building IT usage by category

Issue	Average percentage of use	Prevailing areas of use	Relevance to concurrent engineering
CAD systems Total 3D CAD	 86% ±7% 25% *)	design FM	cooperative work
Database systems	69% ±3%	design construction FM	collaborative work information sharing data integrity and consistency
Knowledge-based systems	< 5% *)	design	collaborative work simulation and forecasting interoperability conflict management
Groupware and workflow management systems	13% ±3%	design	information flow control and monitoring
Document management systems	27% ±4%	design construction FM	collaborative work information sharing responsibilities
Product data management systems	< 5% *)	design	collaborative work information sharing data integrity and consistency responsibilities
Project management tools	48% ±4%	construction	time management resource management control and monitoring
Networking & communication LAN Internet / WWW	 60% ±8% 50% ±5%	design construction FM	information flow and information sharing; communication and process management

*) The amount of analysed data was too small to allow the calculation of a standard deviation.

The following table 2.2 presents an estimation of the interest shown in emerging new technologies by the contacted companies.

Table 2.2: Interest in emerging new technologies

Issue	Companies showing interest	Relevance to this study
Electronic Document Management (EDM) systems	70%	No
Product Data Management (PDM) systems	30%	Yes
Object-oriented databases	30%	Partially
Internet communication tools	26%	Yes
Hypermedia	17%	No
Knowledge-based systems	13%	Partially

At last, fig. 2.4 presents an overview of the estimated benefits to design and construction companies from the use of IT, as seen by end-users and managers of the responding organisations.

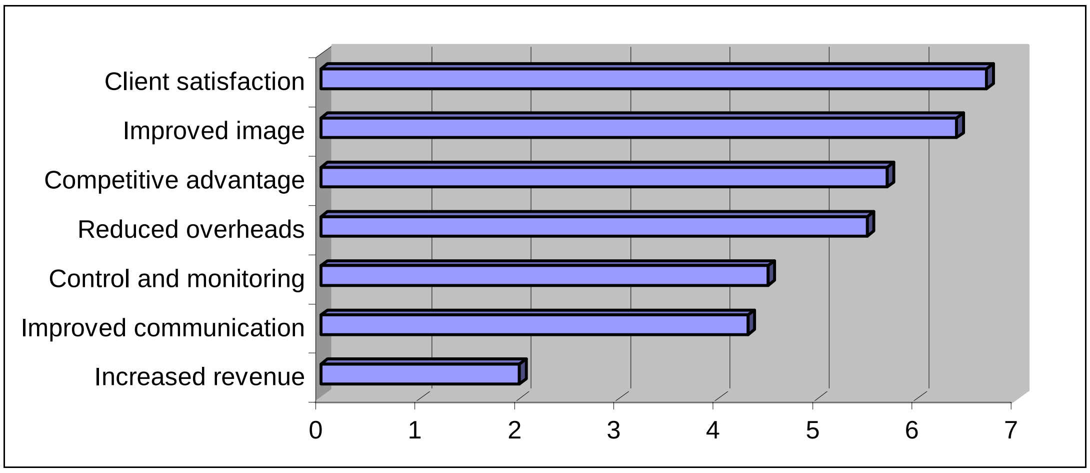


Fig. 2.4: Estimated benefits to design and construction companies from the use of IT (0 = lowest, 10 = highest ranking)

2.2.3 Conclusions

From the analysis of the collected data the following conclusions can be drawn:

1) *Use of CAD and database technology* - Of all types of building IT tools CAD and database systems continue to be most broadly used. CAD systems are becoming more and more sophisticated, absorbing continuously new features from other technologies, like document or workflow management. Besides this, in the last years there is a clearly recognisable shift towards object orientation in place of the existing geometry-oriented models in CAD, and the relational models in DBMS. However, in spite of all achievements, neither CAD nor database systems fulfil the requirements of a framework for concurrent engineering because of the lack of concepts for the treatment of such important issues as conflict management, multidiscipline views, object evolution etc.

2) *Use of PDT* - The penetration of product data technology in building construction is still very low, although it is one of the most important factors for successful implementation of concurrent engineering in practice. Over the past decade different CAD vendors have tried to deploy product models as a kernel for CAD in building design without much success. Unlike the manufacturing industries, product data modelling has not up to now played a significant role in building design, and the practical use of the ISO STEP standard in AEC is still a future issue. However, on-going initiatives like CIMSTEEL (Watson & Crowley 1995) and IAI (IAI 1999a), developing models for CAD and information exchange on the basis of STEP methodology, are likely to bring about the desired shift of paradigm in the next years.

3) *Use of knowledge-based systems* - Knowledge-based solutions are still rarely used, partly because of lack of understanding, and partly because of the disappointment from early attempts in previous years. The benefits of AI-based applications are not yet apparent to designers and contractors, although an increase of interest can be observed (see table 2.2). Here, there seems to exist a strong need for education based on example. However, the modest attempt undertaken in this research work shows that AI techniques can also be embedded in state-of-the-art object-oriented environments to improve the performance of the latter, and at the same time provide a working example of their own usefulness.
4) *Communication and Data Exchange* - The Internet in connection with commercial service providers is becoming the main backbone for communication and data exchange, and Intranet solutions for Local Area Networks already dominate the choice for organising enterprise-wide distributed information access. Comprehensive Internet-based client/server solutions are the vehicle for integrated communication management today.
5) *Interest in concurrent engineering issues* – As shown on fig. 2.4, in the estimation of the application of IT several objectives of concurrent engineering play an important role, such as client satisfaction, improved communication, control and monitoring. Thus, although not always clearly articulated, there seems to be a strong need for reliable computer-integrated environments for concurrent engineering. This is emphasised also by the amount of interest shown in new technologies.

The bottom line here is that many existing IT solutions are ready to be used off-the-shelf and can create enormous change in the quality and efficiency of work in building construction, but only if they can be used together. Sophisticated CAD, 3-D modelling, estimating, scheduling, management, and communications tools all exist today, but they exist to a great extent *in isolation* from each other. What is still missing are logically consistent environments for concurrent engineering processes based on comprehensive and interoperable conceptual models, offering rigorous control of the responsibilities in the coordination, integration and collaboration work within construction projects, along with well-structured Internet-based solutions for effective communication management.

Once again, the major problems that need yet to be solved appear to lie with the conceptual models that should provide the foundation of such coherent environments.

2.3 PDT as Enabling Information Infrastructure for Concurrent Engineering

In spite of its currently modest penetration in building design and construction practice, PDT is best positioned to take over the role of an enabling information infrastructure for concurrent engineering. The benefits of theoretically founded conceptual models are widely acknowledged in the research community, and are gaining increasing attention by the industry as well (cf. Beucke & Ranglack 1993; CIRIA 1995; Howard 1996). In fact, the main reasons for the lack of comprehensive PDT based systems are not in disagreements or disbelief in the basic ideas, but in the lack of standardised, generally accepted models, and sufficiently clarified concepts for the construction of large modelling frameworks spanning over a large number of processes, technologies and application tools*).

The foundation of PDT is provided by the theory of information modelling which originated in the area of database research. Therefore, to gain a better understanding of the capabilities of PDT today, it is useful to look at the development of information modelling as an academic discipline from two perspectives: (1) historically, and (2) from the viewpoint of the categories identified in contemporary research efforts.

2.3.1 Short historical review of information modelling

Information modelling is typically the first stage in the process of developing database applications (Ullman 1988) and is currently acknowledged as the necessary basis for any more sophisticated computer systems (cf. Rumbaugh et al. 1991). Its theory matured in the 1980s but the importance of information models had been recognised already in the 1970s.

Unfortunately, the relationship of conceptual models to engineering design has not been apparent for many years.

In the early days of computer use in the building industry, i.e. in the period **1960-1969**, research and development efforts were concentrated mainly on numerical methods enabling the automation and more precise solution of difficult analysis tasks, such as FEA, optimisation problems, transient heat analysis etc. A few remarkable systems, such as STRESS#) and ICES STRUDL##), featured comprehensive structural models which are still unbeaten w.r.t. many detailed concepts. However, these models were developed without

*) Other industries with large key organisations and respectively larger and better coordinated investments in IT development, such as the automotive, aerospace, process plant and defense industries, have already achieved measurable results in the development and standardisation of large conceptual models, as well as in the implementation of comprehensive PDT based environments. Significant examples are the STEP APs 203 (ISO 10303-203 1994) and 214 (ISO 10303-214 1997) in the automotive industry and AP 227 (ISO 10303-227 2000) in the process plant industry. However, several modelling problems related to the operability of fielded environments are still open issues and one of the main targets of current research and development activities there as well (cf. West & Fowler 1996; ISO 18876-1/-2 2000).

#) Developed by Steve Fenves.

##) Developed at MIT by a team around Robert Logcher.

theoretical modelling basis and suffered from the limited capabilities of the programming languages (FORTRAN) and the computer hardware at that time. Thoughts for relating them to other efforts, e.g. from architecture or mechanical engineering, and to provide for greater flexibility and extensibility of the software had not yet ripened.

In the period **1970-1984** the appearance of CAD systems in the engineering domain and the proliferation of database systems for information storage and retrieval provided new opportunities for computer-supported design work and management.

However, the CAD systems of that time focused primarily on the *results of design*, i.e. the drawing plans. Consequently, they were built around geometry models that enabled efficient performance w.r.t. complex geometry operations but ignored the semantic meaning of the symbols comprising a drawing, leaving that task to human perception. Relationships to emerging database technologies and conceptual models of the *goal of design*, i.e. the building as a product, were not recognised.

In database application, due to the objective of information capture for limited, but versatile tasks in many industry sectors, a more general understanding of the principles of information modelling emerged. Detailed representation concepts were worked out, and a general framework for DBMS realisation was defined. This framework, known as the ANSI/SPARC model (Tsichritzis & Klug 1978), is still valid today. However, in that area too, there were no thoughts yet to extend the technology to support technical development processes.

Towards the **middle of the 1980s** it was recognised that many of the problems experienced by CAD, CIM and CIC were in fact conceptual modelling and database problems (Robinson 1988). At the same time, research in database theory, artificial intelligence and programming languages suggested new powerful representation methods that seemed suitable for the realisation of more ambitious goals. A number of successful pilot projects brought growing interests and optimism that culminated in the inauguration of the ISO STEP standardisation effort in 1984 with the following objectives (cf. Fowler 1995; Owen 1997):

- the creation of a single standard, covering all aspects of CAD/CAM data exchange;
- the implementation and acceptance of this standard by industry, superseding various national and de facto standards and specifications;
- the standardisation of a mechanism for describing product data, throughout the life of a product, and independent of any particular system;
- the separation of the description of product data from its implementation, such that the standard would not only be suitable for neutral file exchange, but also provide the basis for shared project databases, and for long-term archiving.

The optimism of these lines is obvious, especially when we consider that some of the above objectives are far from being reached even today. However, in spite of some problems with its practical application and its overall architecture which shall be addressed later on, STEP contributed enormously to the development of information modelling at least in the following five aspects.

First, it inspired the research community to focus on the definition of conceptual schemas using formalised techniques, rather than prototype development.

Second, it defined a neutral, computer interpretable information modelling language (EXPRESS) which was a significant innovation to conceptual modelling at that time.

Third, it defined a neutral file format for the exchange of data which overcomes many of the limitations of all prior data exchange efforts, such as DXF.

Fourth, it defined a set of conceptual model schemas, the so called Integrated Resources (ISO 10303, parts 41 to 49), covering a broad spectrum of information requirements that are common to all industry sectors.

Last but not least, it promoted PDT as the basis for advanced CAD and information management systems.

Around **1988-1991** first proposals for the principal structuring of the information about building products were made. A number of research papers by Eastman (1988), Turner (1988), Björk (1989), Garrett (1989) and others introduced the product modelling approach into building construction and suggested it as a key technology for construction information exchange and computer integrated construction. Several generic data models, such as the GARM model (Gielingh 1988b) and the BSM model (Turner 1990), intended as input to STEP, the Swedish KBS model (Svennson 1991) etc., developed ideas that are still influential. At the same time, in academic prototype systems, e.g. IBDE (Fenves et al. 1989), DICE (Sriram 1991) and ICADS (Pohl et al. 1992), advanced AI-based representational and computational methods were investigated, and more ambitious requirements and goals were identified.

In the following years, i.e. **1992-1995**, result-oriented projects such as ATLAS, COMBI, COMBINE, PISA etc. (see section 1.3) developed prototype environments that not only proved the principal validity of the PDT approach, but also actively used and improved technologies and methods related to STEP. These efforts brought about better understanding, greater consensus and higher degree of belief in the benefits of PDT. This led to the birth of the IAI in 1995 (cf. IAI 1999a).

Today, PDT is a widely accepted overarching research discipline which addresses many aspects of CAE/CIC including not only data exchange problems but also CAD (Junge et al. 1995a) and engineering databases (Encarnação & Lockemann 1990; Eastman 1992), as well as the construction of large modelling architectures for distributed information management systems for various purposes (cf. Fisher & Froese 1996, Junge & Liebich 1998, Turk et al. 2000).

2.3.2 Contemporary components of PDT in the building construction domain

Research in conceptual modelling in the 1990s has identified several data representation layers of different complexity and scope. Björk (1995) proposes five distinct layers of conceptual modelling efforts that are now generally accepted. However, to reflect better the design of specific large-scale systems, such as CEE, I suggest to enhance Björk's classification by two more categories. This results in the structure presented in fig. 2.5 below.

Björk does not include the innermost and the outermost layers shown in the proposed structure, and mentions that the application layer is too specific and thus not a subject for research.

Other authors provide similar classifications. For example, Hakim (1993) discusses four categories: general-purpose semantic data models, semantic models for engineering design, semantic models for specific design domains, and data schemas for modelling frameworks (the last category is given a different name, but the idea is generally that of a framework); Junge and Liebich (1998) principally agree with Björk but propose to look at the problem from a different perspective and introduce the term “consolidated model” which is similar to the outermost layer in the below figure; STEP adopts (somewhat freely interpreted) the three layers of the ANSI/SPARC model etc.

However, I think that the “7-layer hierarchy” introduced on fig. 2.5 expresses better the fact that each layer depends in a top-down manner on the representations and the decisions undertaken on the previous layers, and that the scope and the level of detail of these layers increases from top to bottom.

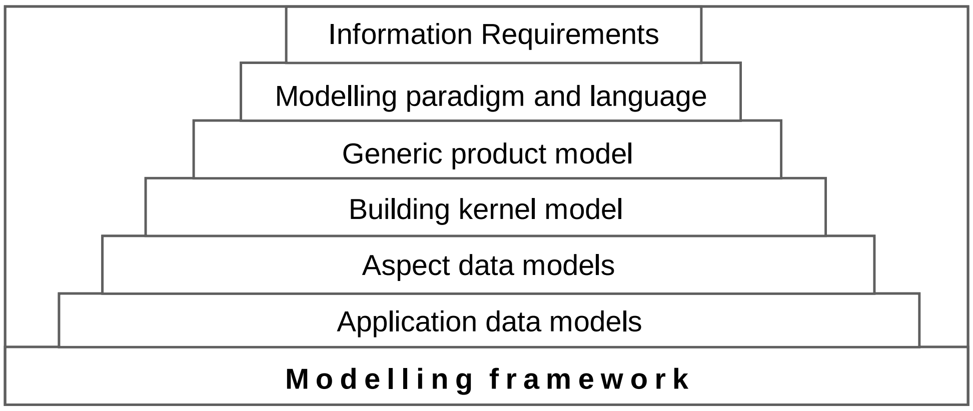


Fig. 2.5: Principal information modelling layers

The **first layer** (information requirements) is the one which is paid least attention as it is usually assumed that requirements are external input to modelling and an obvious development step for any of the other layers. This is principally correct but only part of the truth. Indeed, information requirements of end-users are first acquired and categorised, but they are then “translated” by an information modeller to *specific modelling and IT requirements* which provide the basis for a solution hypothesis and influence the whole modelling approach. Besides this, there exist formal approaches for requirement capture, as well as presentation formalisms such as IDEF0 and SADT. In fact, even if not explicitly recognised as a separate research issue, requirements to engineering data models have attracted the attention of many researchers (cf. Eastman 1993; Scherer & Katranuschkov 1994; Hannus et al. 1995b etc.). Many such efforts have helped to discover problems that have not been identified in “pure” computer science research.

The **second layer** basically deals with the expressiveness of *modelling languages* supporting respective *modelling paradigms*. However, it is important to notice that a language is only a means, whereas the paradigm is a method of representing a UoD. Thus, a language may be good or bad, easy or difficult to learn and use, but it can never show more concepts than defined by the underlying modelling paradigm.

In the present time widest acceptance has gained the object-oriented modelling paradigm (OOM). However, it includes several variations with respect to the treatment of relationships, associations and multiple inheritance which resulted in different non equivalent formal representations such as OMT (Rumbaugh et al. 1991), UML (Rumbaugh et al. 1998), and partially EXPRESS*) (ISO 10303-11 1994).

Earlier modelling paradigms include the relational data model (Codd 1970) and the entity-relationship (ER) model introduced in (Chen 1976) which was also initially adopted in STEP (and de facto silently replaced by the object-oriented paradigm in later years). A good overview of these and other related approaches is provided in (Hakim 1993). They are seldom used for engineering data modelling today because of several deficiencies of their representational capacities.

More advanced representation paradigms originated in the field of AI, such as the frame-based paradigm, the KL-ONE family of terminological knowledge representation etc. Good overviews of such approaches are presented e.g. in (Cunis 1992) and (Russel & Norvig 1995). However, due to the academic character of most efforts, they do not provide appropriate information modelling languages, and are realised in overlays to standard programming languages, most notably LISP and PROLOG.

The **third layer** (generic product model) also deals with high-level modelling concepts but its goals are related to the methods of information structuring for the description of artefacts designed and manufactured by man.

Outstanding efforts for the definition of generic product data models include the “General Architecture, Engineering and Construction Reference Model” (GARM) developed by Wim Gielingh (Gielingh 1988a, b), the “AEC Building Systems Model” (BSM) developed by James Turner (Turner 1990) and the “Engineering Data Model” (EDM) of Charles Eastman (Eastman et al. 1993, 1995a).

GARM uses a unique modelling approach organising construction project information at a very high level of abstraction. Its key idea is featured by the abstract object called *Product Definition Unit* (PDU) which is intended to be specialised (at different times!) to its subtypes *Functional Unit* (FU), representing the data *as designed*, and *Technical Description* (TS), representing product data *as required*. Originally, GARM proposed a classification of seven subtypes (as required, as designed, as planned, as built, as used, as altered and as demolished) which emphasised the life cycle stages and the different representation aspects undergone by building products. As a whole, it is a very general model which is not appropriate for short and medium term implementation, and thus was not accepted by STEP. On another side, it is the first model that recognised the relationship between top-down design processes, with the designers endeavouring to satisfy functional requirements, and bottom-up construction realisation, with product manufacturers providing their products as technical solutions for such requirements. This idea has been further developed on more pragmatic level in later modelling efforts such as the proposed “Realization Model” in (Wittenoom 1997).

*) EXPRESS is in fact not a pure object-oriented modelling language as it does not include some recognised object-oriented concepts like polymorphism and encapsulation. However, on data level, it provides sufficient constructs to create comprehensive object-oriented models. It supports modelling principles like classification, abstraction, aggregation and inheritance which form the basis of such models. With some additional flavours, as proposed in the PISA project (Braun et al. 1994), EXPRESS fulfils most of the requirements related to the modelling effort attempted in this thesis.

In contrast to GARM, BSM deals explicitly with buildings and adopts a top-down systems approach by which the separate functional systems that make up a building (spatial, structural, mechanical etc.) are also separately modelled, and are supposed to be “glued” together by a high-level AEC building systems model. This approach is oriented towards the functions of building parts, and not the technical products or materials chosen. It appeared to be more natural to designer thinking and was one of the main precursors of the kernel-aspect model architectures.

The core of EDM[*)] is situated somewhere between the second and the third layer. It uses sets and first-order logic and is capable of modelling both relationships between objects and value constraints to object attributes. The representation is based on a small number of primitive constructs (domain, aggregation and constraint) which are used to build high level forms modelling the actual building semantics. A *domain* is defined by a name and a set of possible values (attributes). *Aggregations* are composed of variable-domain pairs, where a variable may have zero or more values including aggregations and domains. A *constraint* is defined by a name, a predicate expression and a list of variables occurring in the expression. The high level forms include functional entities, accumulations and compositions. *Functional entities* are roughly equivalent to the usual meaning of objects but differ in the treatment of specialisation in that both classes and instances can be specialised. *Accumulations* provide 1:N relationships between the properties of a composite object and its parts and enable more explicit treatment of “part-of” links than conventional object-oriented approaches. At last, *compositions* define sets of accumulations that all apply to the same set of functional entities.

As a whole, EDM overcomes many known difficulties in the development of engineering data models. In contrast to the data-centric approach of STEP and IAI, it supports both design data representations and functional features. However, although it proposes a comprehensive framework, EDM is not sufficiently modular which provides some concerns about the flexibility and scalability of the approach.

Another interesting development similar to EDM w.r.t. the modelling scope is SHADES (Hakim 1993). It is based on the description logic paradigm and provides an object-centred approach which solves many of the problems of object-oriented models. SHADES supports object evolution, schema evolution, incomplete representations and dynamic recognition and classification of objects. It is capable to accommodate several hierarchies of design aspects within the same environment, automatically enabling their interoperability. However, it provides similar concerns w.r.t. flexibility and scalability as EDM.

The **fourth layer** (building kernel model) attracted great attention in the last years when it was recognised that a successful definition of an adequate kernel may be the key to success for the modelling of large-scale IT environments. The idea emerged bottom-up, from the development and successful prototype implementation of several discipline-specific aspect data models. Detailed examinations of these models showed that if they are developed independently and without commitment to common higher level concepts, there is little hope that any reasonable link among the model worlds would ever be achieved.

[*)] EDM proposes a complete environment, based on a set of core concepts, but there are no distinct names of the separate layers of the EDM framework.

The purpose of the kernel data model is threesome: (1) interpretation of common requirements, (2) specification of common data, and (3) development of a common framework (Wix 1996). From that point of view, a framework should be understood as a consistent basis for the development of more specific domain data models, and not as the consolidated model structure of an implemented IT environment.

However, whilst the purpose of a kernel data model seems to be quite clear, its content is not. There are at least *three distinct strategies for the design of a kernel model schema*: (1) top-down definition of constructs capturing basic relationships, expected to be supported by all domain-specific data models, (2) bottom-up integration of overlapping domain data, and (3) autonomous domain model schemas linked through a kernel that does not define common objects classes, but a common communication paradigm for the software tools implementing the domain schemas.

The first strategy follows the idea of the BSM approach and assumes a *pre-harmonised* model world.

The second strategy is similar to the principal methodology of constructing multidatabases (cf. Kim 1995; Dadam 1996) and assumes a *homogeneous* model world.

The third strategy is related to distributed agent systems and assumes a *heterogeneous* model world, along with intelligent agents capable of “talking” to each other.

A very popular variation of the first strategy is the so called minimal approach introduced in (de Vries 1991). Its key objective is to keep the kernel extremely small in order to allow for greatest possible flexibility and adaptability to different domain needs. Thus, different types of building elements are not modelled as separate classes, but are included as the values of a building component type attribute of a generic building component class. A more pragmatic realisation of this idea is provided by the IFC Kernel Model (IAI 1999a, b). It adopts in principle the minimal approach but extends the number of kernel objects with additional constructs, such that envisioned software implementations would be less difficult to achieve.

The second strategy is suggested by STEP where every domain data model can only use and further constrain definitions already provided in the Integrated Resources. It has been attempted e.g. in the ATLAS project, but the invested modelling efforts have shown that the definition of a general kernel model meeting the requirements of ISO STEP is a difficult task entailing a huge amount of resources, and mapping to discipline-specific models is nevertheless unavoidable. Therefore, unlike other industries, the acceptance of this strategy in building construction is at best moderate.

The third strategy has been explored in academic research in the AI area, e.g. in the ACL/KIF approach (Khedro et al. 1994). It has been successfully applied in practical systems of limited scope and well-defined narrow goals, as e.g. in the ARCHON project, as well as in several subareas of computer-aided manufacturing (cf. Müller 1993). However, due to the complexity of the implementation and the strong requirements to application tools, it has not been tried on a more general level.

A mixed strategy of (1) and (2) seems to be implemented by the COMBINE project, and a strategy combining (1) and (3) has been explored in the COMBI project.

The **fifth layer** (aspect data models) addresses the definition of information structures for particular disciplines and/or phases of building construction*).

Aspect (or domain) data models typically inherit or copy many information constructs of the kernel model layer, but there are also other possible approaches as demonstrated in SHADES and, partially, in COMBI. Such data models are often developed within the frames of larger projects for several domains at a time, as e.g. in the ATLAS and the COMBINE projects. However, there are also successful “self-contained” realisations with very specific goals and without consideration of potential overarching environments requiring kernel model constructions. Examples of such efforts include the CIMSTEEL model (Watson & Crowley 1995) and the German structural steel model originally proposed in (Haller 1994).

The **sixth layer** (application data models) includes the explicit or implicit conceptual schemas implemented in individual software applications. It is generally excluded from research and standardisation efforts because of the low possibilities to reuse such schemas.

However, there are two different situations to be considered here.

In the first case, the application is typically developed when an appropriate aspect data model is already available, and its conceptual schema can easily be adapted to that aspect data model by means of standard object-oriented techniques.

In the second case, the application has been developed *before* the aspect data model, and its native data structures, often utilising a very different paradigm, cannot be changed. This is the more difficult and, unfortunately, more frequent situation to date. The related problems are insufficiently investigated because it is generally assumed that the necessary model transformations can be accomplished only by specific conversion software which is on the responsibility of the application providers. However, whilst there is certainly very little that can be done for the harmonisation of the data, it is possible to implement mechanisms that can help to conceptualise and automate the mapping process. Such efforts have been undertaken e.g. in the COMBI project (cf. Ammermann et al. 1994).

The last, **seventh layer** comprises the modelling framework. Many researchers define it as the totality of the other already discussed layers and thus do not see a necessity to examine it as a separate category. I think that it has its own principles and design methods that need to be considered separately.

The specific issues addressed at the modelling framework layer encompass:

1) the strategy for achieving the interoperability of the other model layers,
2) representation constructs enabling the realisation of this strategy in general, and
3) implementation forms and methods enabling its realisation in particular, i.e. in a practical IT environment.

The largest initiatives developing principal concepts for the construction of comprehensive modelling frameworks relevant to AEC are ISO STEP and IAI/IFC. They are shortly discussed in Appendix VII, along with the various specific reference models used for validation of the concepts developed in this thesis.

*) In the ATLAS project the name “view type models” is used, whereas COMBI defines “partial models”, and IAI speaks about “domain models”, but the idea is generally the same. Whilst in this thesis I mostly use the term “domain (data) model”, in this particular section “aspect model” is preferred to preserve the terminology provided by Björk (1995).

Especially the IFC framework, which closely follows the principal information modelling layers shown on fig. 2.5, appears to be well suited for the envisaged concurrent engineering environment even if there are no specific hints in that respect in the IAI documentation.

2.3.3 PDT application in other industry branches

In chapter 1 it was mentioned that other industries have several advantages w.r.t. the development and the practical realisation of environments for concurrent engineering, mainly due to the existence of global key organisations playing the role of research and technology drivers. Therefore, without going into a detailed analysis that might take dozens of pages, it is interesting to examine major efforts in other industry sectors and compare these to building construction.

Characteristic for the **automotive**, **aerospace** and **defense** industries is the development of large-scale systems built around large, tightly integrated modelling frameworks. Such frameworks typically guarantee a high level of internal consistency, whereas external inter-model operability is mostly limited to file-based data exchange accomplished with the help of dedicated sophisticated translators.

An outstanding example of a commercially available, widely used system of that kind is METAPHASE (SDRC 2000). It provides specific solutions for each of the above-mentioned industries by incorporating a number of capabilities to support concurrent engineering processes, such as product data management, configuration management, change management, scheduling, systems engineering and requirements management, along with interfaces for design collaboration to major CAD systems (I-DEAS, CATIA, AutoCAD) and the STEP AP 203. However, just as other similar systems, its conceptual design does not provide for an easy method to adapt the system's features to the needs of the AEC sector.

Research and development work in the area of conceptual modelling and standardisation basically adopts the STEP harmonisation strategy aiming at the specification of fully homogeneous and consistent data models. The results of such activities include the STEP APs 203 (ISO 10303-203 1994) and 214 (ISO 10303-214 1997) documented on several thousand pages - an unreachable effort for other domains.

Software implementations are concentrated on the application of suitable methods for multidatabase integration and distributed object-oriented processing using CORBA technology. However, whilst such technical solutions are in principle valid for building construction as well, the amount of work needed makes them practically not applicable.

More interesting are therefore the achievements in industry sectors that are closer to building construction w.r.t. their general economic and technical parameters, such as the **shipbuilding** and the **process** industries.

Intensive research and development efforts in the shipbuilding domain, in conjunction with a set of international projects started with the European MARITIME project (de Brujin et al. 1995), have led to a unique solution approach, known as the "MARITIME AP Factory", which takes an intermediate way between the "pure" STEP approach of strictly harmonised models and the pre-harmonisation idea followed by the IAI (cf. Wix 1996). Its essence is in the use of *generic templates* as building blocks for AP development which capture not only representational but also behavioural and reactive features. The definition of such templates is accomplished with the help of a dedicated modelling language, GEM, that incorporates both activity and data modelling constructs. Model integration and interoperability basically follow the STEP approach but enable tighter coordination of process and product information and take in consideration several system implementation features. Thus, in contrast to STEP APs, the MARITIME models are meant to be "directly implementable" in IT environments.

The efforts associated with the development of such models also seem to exceed the limited resources that can be afforded in building construction but the idea of using meta data templates for the construction of more complex semantic concepts provides interesting possibilities that are worth considering. In fact, it is to a certain extent analogous to the idea of using *mapping patterns* proposed in this thesis, which will be presented in chapter 5.

Most closely related to the conceptual modelling approach adopted in building construction is the development work in the process plant industry sector, carried out in conjunction with the POSC/CAESAR initiative in the frames of the EPISTLE project (Angus & Dziulka 1998). The undertaken efforts draw upon previous work by Gielingh (Gielingh 1988a, b) and West (West 1994) and have culminated in a standardisation proposal (ISO 18876-1/-2 2000) for the construction of a comprehensive *integration model* of general applicability, which has great similarity to the strategy of the IAI. In this proposal, known under the name IIDEAS, along with the specification of a meta model and a core model, greatest attention is paid to the mapping methodology needed for bi-directional data transformations between the integration model and application-specific data models, as well as to the problems related to data consolidation. To tackle these problems, an *integration procedure* is proposed whose essence is in updating the integration model in such way, as to allow the unambiguous derivation of an integration model subset, or application view model, that can then easily be mapped to/from the application-specific data model at run-time. This procedure is schematically illustrated on fig. 2.6 below.

Thus, in the process industry as well, the efforts for the implementation of interoperable environments are mainly focused on techniques related to database integration, requiring considerable implementation efforts.

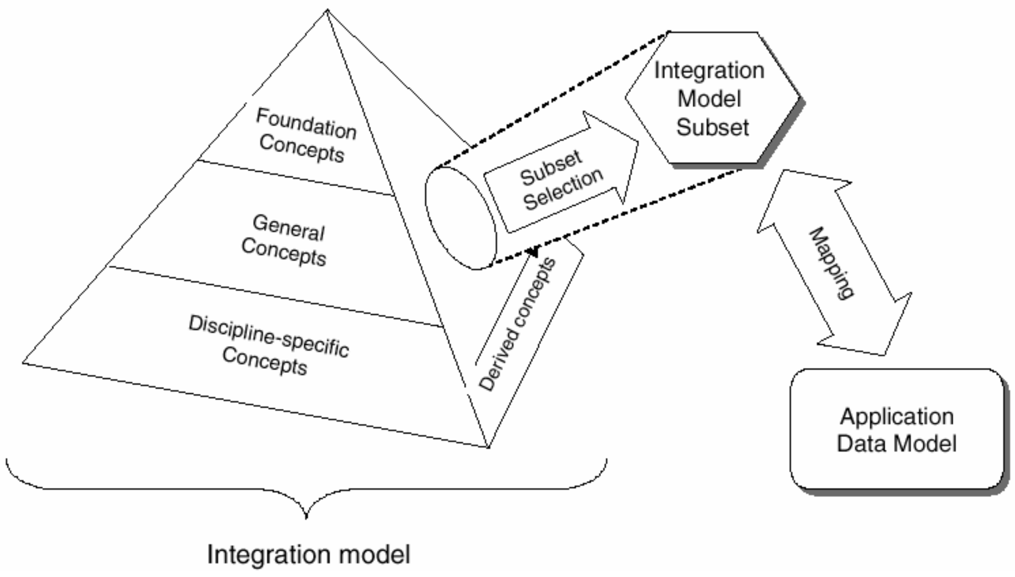


Fig. 2.6: The IIDEAS integration process /after (ISO 18876-1/-2 2000)/

2.4 Benefits of the Use of a PDT Based Framework for CEE

The methods used for the representation of construction project processes and data have evolved over centuries, shaped by the efforts of designers and contractors to find a suitable format for the coordination of decisions and descriptions about the constructed facilities. Until recently, this suitable format was that of documents on paper.

Today, with the existing capabilities for digital processing of electronic documents, the efficiency of document production and the effectiveness of their distribution and co-ordination are enormously improved. However, taken alone, this kind of information representation is no longer adequate for the complex production processes and the increased customer demands of the present time. It does not guarantee completeness and consistency of the project data and it does not provide an objective model of the product of design. Thus, before a building is erected, it is only a virtual collection of incomplete "mind models" of the involved professionals, only partially supported by projections of these models in reports and drawing plans. The process of *model realisation* is insufficiently coordinated and bears many unexpected risks. As a consequence, concurrent engineering cannot be efficiently practised.

With the use of information systems based on PDT such problems can be greatly overcome because:

1) A PDT based system would enable the modelling of all types of technical data explicitly, by using unified and logically consistent representation methods, and not implicitly, through textual descriptions and drawing;
2) It can cover the information needs of a variety of IT tools due to the availability of rich semantic concepts as compared to the limited representation capabilities of document and geometry models;
3) It can enable the integration of common data across disciplines and applications at more detailed levels than a document based approach;
4) It can provide just-in-time information in appropriate presentation formats due to the capabilities for clear separation of semantic and presentation features;
5) It can facilitate the coordination of decisions, processes and data at any time because of the constantly maintained up-to-date state of the model data.

A comprehensive modelling framework, reflecting the architecture suggested on fig. 2.5, would be able to provide all the above features. Of course, the construction of such a framework is not an easy task. The move from document-based design to model-based project realisation cannot be expected to happen in only a few years, but the progress made with ISO STEP, IAI/IFC and the many successful research and development projects of the last years shows that a paradigm shift is under way.

However, STEP currently suggests only a very general methodology, and provides little hints as to how it should be implemented. On the other hand, this methodology obliges possible STEP-based environments to follow a more or less pre-defined schema.

IAI/IFC provides a set of specific modelling principles which should guarantee the high level of harmonisation of the existing (and future) components of the framework, but does not define any enabling methods or supporting conceptual schemas w.r.t. the operability of IFC-based concurrent engineering product development environments.

Exactly these missing features are addressed in the interoperability methods for CEE proposed in this thesis.

Chapter 3: Outline of the Proposed Concurrent Engineering Environment

Jack considers **M** *to be a good model of* **A** *to the extent that he finds* **M** *useful for answering questions about* **A** .

– Marvin Minsky, The Society of Mind

This chapter introduces an environment in which the model mapping approach developed in this thesis can most efficiently be applied. Presented are the principal features of an IT system for concurrent engineering that goes beyond looking at the technical products, processes, documents and information to observe the environment as a whole, including the people, the technology and the information management issues involved. On the basis of outlined specific concurrent engineering requirements to PDT and an anticipated multi-tier client/server architecture, a wider framework for *environment modelling* is proposed. It comprises not only the representation of the product data describing a construction facility, but includes also the modelling of system related aspects, such as actors, roles, access rights etc., and the modelling of the information about the information system itself (clients, servers, communication model). Suggested is a hierarchical architecture comprised of a high-level meta model, a set of layered product data models, and a set of models and methods supporting the operability of the environment.

The emphasis of the proposed approach is *not* on the modelling of the “objective reality” of construction products and processes themselves, but on the interoperability of the envisaged software system. Therefore, requirements and problems related to “pure” product modelling are only briefly covered (as it is assumed that the models developed by the IAI and STEP will soon penetrate into building construction practice), but the *interoperability problems* related to the environment are discussed in depth, and a modular approach for their solution is outlined. The latter forms the core of the developed concepts, presented in detail in the following four chapters of the thesis.

3.1 Preliminary Remark

The development of a CEE system is hardly a task for one person. Its design is a complex process which involves simultaneous consideration of many cross-domain aspects, such as (1) the general requirements and characteristics of the target industry area, (2) end-user requirements of the relevant engineering disciplines, (3) technical requirements to the features of the models and tasks that need to be supported, (4) operability requirements w. r. t. data management, consistency, system interfaces, performance etc. Practical software realisation adds aspects like available technology, requirements and constraints of existing applications, adequate consideration of relevant data processing standards, hardware and software resources and so on. Typical for such broad systems are design decisions that cannot be deduced analytically. Also typical is the possibility of different solution approaches, emphasising more strongly one or another aspect of the set up overall goals.

The concepts of the CEE system outlined in this chapter have been refined gradually, taking into account past cooperative efforts undertaken in the COMBI and the ToCEE projects. The aims of these projects have been somewhat different, but the general vision has been largely the same, i.e.:

1) To provide a flexible framework capable to support a wide range of engineering applications;
2) To enable the use of not fully harmonised data models, taking into account the heterogeneous, fragmented and multidisciplinary nature of AEC, and
3) To enable independent work on local model views as appropriate for the different discipline-specific tasks involved in the building design process.

These envisaged features have been used as baseline for the decisions taken during the conceptual design of the CEE system suggested herein. It is a generalisation of the environments developed in COMBI and ToCEE, with special emphasis on a number of interoperability problems that have not been investigated in these projects. Its architecture and the underlying modelling framework set up the stage for a deeper discussion of the developed model mapping approach for concurrent engineering processes, which is the main subject of this thesis.

However, many of the addressed interoperability aspects are not specific for the proposed CEE system, but are also relevant for other possible system architectures (cf. Böhms & Storer 1994; Amar et al. 1997; Thomson 1998; Crowley & Watson 2000; ISO 18876-1/-2 2000). Therefore, for the overall objectives of the research, the CEE system itself is in fact a subordinate issue. In accordance with that, its presentation is kept concise, and most requirements and design decisions are reported without going into the specific details of why and how they have come about. Some critical issues are discussed in section 3.8 at the end of this chapter. More details are provided in (Karstila, Katranuschkov & Mangini 1996) and in (Scherer 1997a).

3.2 Requirements to the CEE System

In chapter 2, the basic features of the concurrent engineering methodology and their relevance to building construction practice were identified. In the discussion of these features a few broadly formulated requirements to information technology could be identified. However, as pointed out in section 2.3.2, in order to develop a successful concept for a comprehensive CEE system, these informal requirements have to be synthesised to specific conceptual, software development and performance requirements related to the information system as such.

This synthesis has led to a set of specific IT-related requirements which are summarised in table 3.1 below.

By examining this table, it can be noticed that integrated project data models, which have been the main purpose of many development efforts in the area of PDT, are an important, yet not the only data set, nor the only conceptual issue required to solve the problems related to concurrent work.

Therefore, for the successful application of an IT system for concurrent engineering it is necessary to observe all related system components as coherent parts of a much wider framework for **environment modelling**. The basic characteristics of this framework and a principal approach for its construction are presented in the following sections.

Table 3.1: Concurrent engineering features and their implications to model-based project data management

Concurrent engineering features	Project data management issues	Relevance to this study
Collaborative work of distributed design teams	• Integrated data model • Common product data repositories • Distributed client/server environment	+ + +
Concurrency	• Concurrent multi-user access to the product data repositories • Availability of local data allowing independent work in the individual engineering domains • Change management and versioning	+ + +
Efficient inter-discipline communication and information sharing	• Flexible client/server interfaces • Comprehensive set of available server functions • Consistent support of interoperability aspects	+ + +
Life cycle management	• Comprehensive set of harmonised data models • Modelling representation enabling model evolution and different levels of repr. detail	- (+)
Robust project management	• Capturing of key project management data in appropriate modelling objects • Linking to process and workflow data	- -
Consideration of responsibilities and access rights	• Capturing of legal issues like contracts, order assignments, approvals, access rights etc. • Appropriate linking to documents, the main holders of legally binding data	- -
Simulation, monitoring and forecasting	• Adequate interfaces and mapping facilities to dedicated advanced software tools	(+)

3.3 Basic Design Principles

Within a CEE system, basically three types of information are generated and manipulated (cf. Turk et al. 1997):

1) *The information about the constructed facilities*, including construction products, construction processes, construction codes, regulations and requirements etc. Information in this section relates to real-world material objects and is not associated or influenced by IT. For example, when we model the reinforcement of building elements, we are not necessarily interested of how this information is stored in documents, databases, or how it is versioned, updated, accessed and so on.

2) *The information about the CEE system itself*, including information products (documents, databases), information processes (tasks, activities), and the environment as such (hardware, software, servers, clients, persons, applications). Concepts of this type are largely generic. They are related to construction concepts, but their discriminating feature is domain independence, because their actual domain is the IT system itself, and not any of the physical domains it supports.

3) *The information about the information itself*, i.e. concepts like information ownership, authorship, versioning, and even the more abstract concept of a model as such.

A CEE system stores information as CEE data. This data can be stored in various kinds of databases, such as relational or object databases, or simply in a flat file database.

The types of CEE data are defined in a CEE schema. This schema is comprised of several, not necessarily harmonised component schemas, fitting into a CEE *modelling framework* (CEEMF). All these components must be "somehow" inter-related; hence the singular "schema", and not the plural "schemas" is used for CEEMF. The challenge here is to define what does this "somehow" actually mean.

The modelling framework (CEEMF) must define how to decompose the CEE information, i.e. it defines the decomposition principles, the decomposition itself, the main features of each component and the general interfaces between the components, which should enable the overall operability of CEE.

However, before looking into how to organise the CEE system information, some basic decomposition criteria need to be defined. These criteria are suggested as follows:

- *Simplification*
  The decomposition should simplify the modelling and consequently the implementation of CEE without sacrificing features required for its functionality.
- *Autonomy*
  The details of each component should be largely autonomous, so that they can be developed and implemented with little reference to other components.
- *Domain knowledge separation*
  The decomposition should "follow" knowledge domains. Components should separate general knowledge from domain-specific knowledge, so that experts in each field can easily work in an encapsulated context of their domain expertise.
- *Rich intra-component relationships*
  The information within a component should comprise a rich set of reference types (uses, used by, specialised, part-of, is-a, contains etc.) for the representation of the relationships within the component.
- *Lean inter-component relationships*
  The information within each component should have as few as possible external references to other components so that data inter-dependencies and data transformations, as well as the overall consistency of the full set of component models would be easier to manage. This principle is in fact broadly analogous to the method of substructuring mechanical systems in structural engineering. It is not of much importance for a system where fully consistent harmonised data models are available, but it can greatly facilitate mapping tasks in non homogeneous systems where this is not the case.

- *"Use" instead of "include"*
  The components should "use" rather than "include" knowledge which is not component specific. For example, products need not know about the properties of the documents in which their own properties are presented or archived, but if necessary can "use" these documents to service a "viewing" request. Such relationships can be achieved by the concept of "delegation". Deep "is-a" hierarchies are undesirable, because, at the implementation level, they tend to bind more tightly than delegation. The rationale for this principle is similar to the previous one, but the aim is to facilitate interoperability at software component level and not at the level of semantic modelling.

The decomposition of CEEMF follows the general idea of *duality of information processes and construction processes* (Scherer 1997a), which is expressively illustrated in the high-level reference model of the general CEE system process suggested in (Björk 1999).

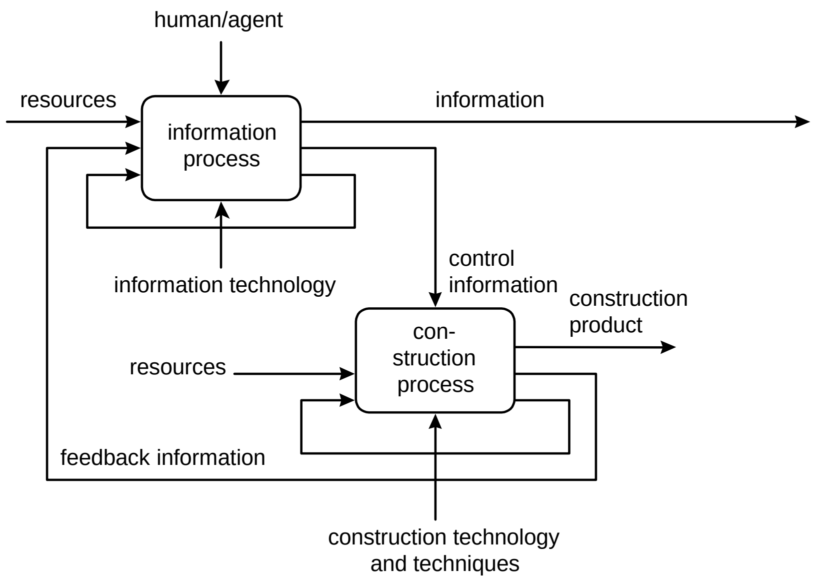


Fig. 3.1: High-level reference process model of the CEE system in IDEF0 / after (Björk 1999) /

In the explanation to his original figure, Björk uses the abstraction of material processes and information processes. Material processes change the material world (e.g. pouring concrete into a formwork), whereas information processes process the information about these material processes (e.g. how much concrete is needed, when and by whom). Information processes control the material processes and monitor their output, and the material processes provide feedback data to the information processes.

Following this observation, a CEE system for building design can be seen as a generalised information process that supports, with the help of IT, both the information and the "material" design activities. In accordance with that, a principal *decomposition matrix* for CEEMF can be defined.

Table 3.2 shows the proposed matrix "populated" with some sample objects. It is divided row-wise into construction specific information, CEE system information and generic concepts.

Construction information is further decomposed into product, process, document and legal (or requirement) information. Similarly, CEE system information is decomposed into CEE system product (which is information) and CEE system process (during which this information is created or modified).

Table 3.2: Modelling framework decomposition

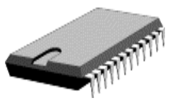
Information type	Information subtype	Sample concepts	Further decomposition
Construction information	Product information	Project, building, storey, site, element, connector, space, wall, slab, column, equipment, opening, door, window, material, ...	By specialisation, using the systems approach (kernel, aspect, application models)
	Process / activity information	Construction process, person, organisation, work schedule, work work section, work group, work task, ...	By specialisation and by building life cycle stages (brief, scheme design, tender, construction, ...)
	Documents	Bill of quantities, cost schedule, drawing plan, request for information, contract, ...	
	Legal data	Regulation, requirement, clause, provision, authorisation, ...	
CEE system information	Product information	Information container, repository, file, message, generic document ...	
	Process / activity information	Client, server, actor, user, application, communication event, transaction, request, response, input, output, ...	
	Legal data	Access rights, approval, authentication, notification of receipt, digital signature, ...	
Generic information 	Meta information	Owner, lock, version, status, configuration, view, ...	
	Object model	Model, schema, class, object, attribute, relation, operation, ...	

Fig. 3.2 shows a respective high-level reference model of CEE system information, derived from fig. 3.1 and the decomposition principles outlined above.

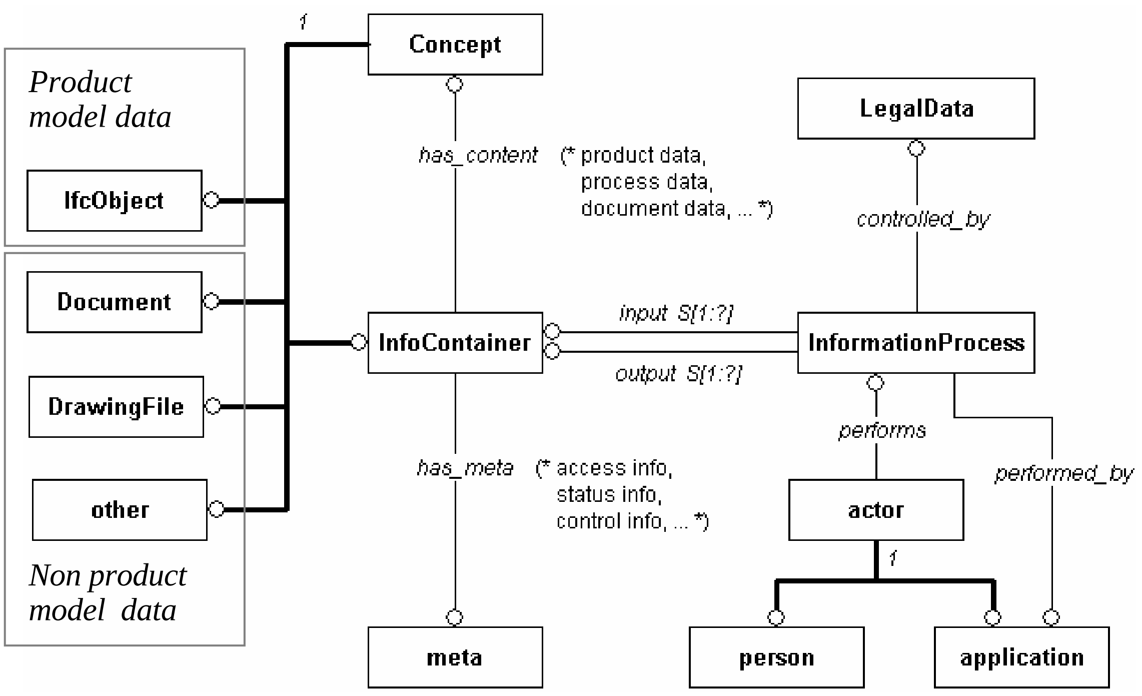


Fig. 3.2: High-level reference information model of the CEE system in EXPRESS-G

The important output of the information processes (represented in this perspective as information items themselves) are *information container objects*.

An **Information Container** can be generally defined as “the smallest chunk of data with its own meta information”. In this general treatment of the information in the CEE system, an analogy to the “knowledge chunks” proposed in the area of artificial intelligence (cf. Newell 1982; Cunis 1992) can be observed.

The data in an Information Container is about construction products or construction processes which fit into the schemas related to the first row in table 3.2. This data is associated with meta information which is about who, when, how created the Information Container and how it is related to other data in the framework.

Examples of Information Containers include high-level concepts like documents, or even a whole model as such, but also simple entities or attributes of entities. Information Containers are both the input and output of information processes which are performed by actors who may be persons or applications.

In a distributed CEE system, Information Containers are the input and output of client/server interactions, and these interactions themselves (requests, responses, transactions). However, Information Containers provide only a high level of representation that does not tell much about the particular content. Their main purpose is to provide a uniform method for *information transport*, and not to support domain/application semantics. Therefore, to be able to satisfy the information requirements of specific design tools, a structured **modelling framework** is needed, enabling different representational levels of detail.

3.4 Modelling Framework

Fig. 3.3 presents a proposed principal modelling framework for CEE developed by the author together with R. J. Scherer as part of the research work performed in the projects COMBI (Katranuschkov 1995; Scherer 1995) and ToCEE (Katranuschkov & Scherer 1997; Hyvärinen, Katranuschkov & Scherer 1997; Katranuschkov & Hyvärinen 1998). It is characterised by the following features:

1) It corresponds to the design principles identified in the previous section;
2) It provides a broad scope of information types that have to be dealt with in a CEE system;
3) Its structure is almost fully compatible with the IFC modelling architecture (which allows to examine the applicability of the IFC models for CEE by performing only a few additional tests, and not re-implementing the full model structures);
4) Unlike the currently only available STEP AP in the area of building construction (ISO 10303-225 IS 1999), it is *not* fully harmonised and therefore allows the investigation of interoperability issues related to a distributed, non homogeneous model world, typical for the distributed, highly-fractured domain of AEC (cf. IAI 1999a);

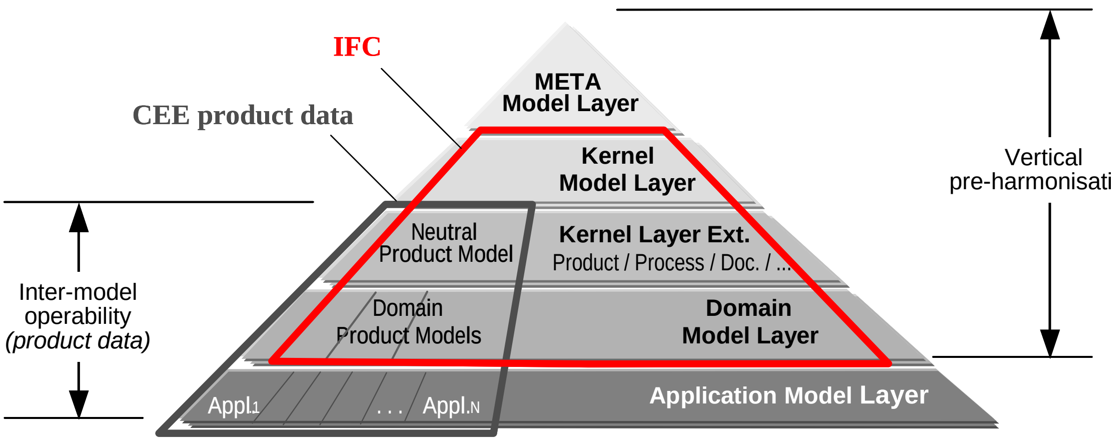


Fig. 3.3: Reference architecture of the modelling framework

The suggested “implementable” framework is structured in five hierarchical layers as follows:

- The **Meta** model layer defines explicitly the basic principles of the modelling paradigm with system-wide applicability.
- The **Kernel** model layer defines high-level generic concepts which are common to all lower level models representing product, process and document related information.
- The **Neutral** model layer extends the kernel layer by defining high-level concepts for each modelling perspective, i.e. Neutral Product Model, Neutral Process Model, Neutral Document Model etc.

- The **Domain** model layer further specialises the neutral model layer. However, because of the different granularity of product, document and process data, domain models currently exist only for product related information. This layer corresponds to the systems approach of the “AEC Building Systems Model” (Turner 1990), subdividing the model domain according to the different domain aspects of design (architectural, structural, HVAC, and so on).
- At last, the **Application** model layer contains the native models of the applications to be used in the CEE system. Unlike the models on the other layers, which can be standardised and, at least hypothetically, fully harmonised, the application models will always remain “external” to the other parts of the framework, because the scope and number of the application systems to be integrated in CEE is unpredictable in advance (Hannus et al. 1995b). Therefore, at least at that level, the tackling of related mapping problems needs to be considered.

With respect to the operability of the envisaged overall environment, greatest interest provides the Meta model layer. It is the component that “glues” together the system by representing explicitly the rules by which the other model layers are constructed, and not implicitly, by means of an external to the system modelling language paradigm.

The principal structure of the Meta model and its explicit usage by the lower levels of the framework is shown schematically on fig. 3.4.

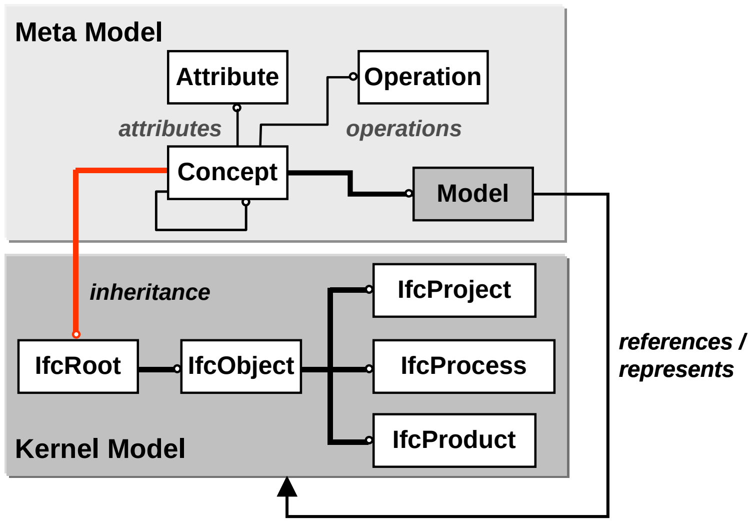


Fig. 3.4: Principal structure of the Meta model and its relationship to the lower layers of the modelling framework

Its main features are as follows:

1) It provides a basic **Concept** class which defines not only attributes but also operations, allowing to add functionality to the data models;
2) It provides a **Model** class which defines the meta data associated both with data model schemas and instantiated models, which enables their treatment as a whole, by dedicated model-level operations.

These features of the Meta model are of great importance for the overall interoperability in the CEE system. They are discussed in more detail in the next section and in chapter 4.

3.5 System Architecture

In accordance with the requirements for collaborative work of distributed project teams, the choice of an Internet-enabled *client/server system* is proposed.

Indeed, client/server solutions making full use of the Internet are state of the art for distributed project development processes, as are typical for the virtual enterprise in building construction. However, there are quite a few different architectures that can be envisioned for a concurrent engineering environment based on the client/server model.

On the client side a variety of tools have to be supported, such as:

– Legacy application with embedded Internet functionality (fat client),
– Stand-alone legacy application – Client adapter (a variation of the above fat client),
– "Thin client" application, relying heavily on centralised product data services,
– On-line services, such as Applets embedded in a WWW-Browser etc.

On the server side, the following alternatives can be envisaged:

1) *All-in-one Project Data Server*
This is, conceptually, the simplest possible server design. It has the benefits, but also the drawbacks of not being dependent on any other services. To satisfy the requirements of concurrent engineering, the server implementations must embed many features that go far beyond pure data management functionality, such as process and workflow management, actor authentication etc. These features can make the server architecture difficult to achieve and insufficiently flexible for future enhancements.

2) *Object Request Broker – Project Data Server*
In this approach, the Object Request Broker takes over all communication related tasks, and the Project Data Server takes over the data management tasks. By modularising the architecture in this way, a conceptually clear extensible structure can be accomplished. Moreover, a set of dedicated servers, e.g. for document, process or regulation management could easily be "plugged-in", providing the features that are not in the scope of product data management.

3) *Object Request Broker – Project Data Server – Project Data Agents*
This enhancement of the above architecture, shown schematically on fig. 3.5 below, provides a further level of flexibility, allowing the integration of intelligent server-side agents capable of accomplishing sophisticated data management tasks. Such tasks include: matching of two or more model versions, consistency checking or code checking services, or even more complex server-side processes such as conflict management. In addition, by specifying a common agent-server communication paradigm, it would even be possible to implement knowledge-based agents "wrapped around" a traditional database server using relational or object-oriented technologies.

I consider the last of these alternatives as the most appropriate for the needs of CEE because:

1) It enables the coherent use of traditional object-oriented methods along with symbolic and rule-based processing for the tackling of complex data management tasks, which is of utmost importance for the realisation of sophisticated model transformation methods and advanced view generation mechanisms.

2) It ensures that different types of engineering applications using different representation paradigms can be plugged into the system because the details of the server functionality need not be known to the clients.
3) It provides a high level of modularity since agents can encapsulate appropriate autonomous actions. Moreover, such actions can be executed automatically and concurrently, in dependence of the actual state of the data in the common project data repository of the system. In this way, different consistency problems can be solved more efficiently, server response times can be improved, and a more “intelligent” behaviour of the system can be achieved.
4) Last but not least, it enables the realisation of cooperative problem solving methods. Whilst such issues have not been in the scope of this study, in a comprehensive CEE system various tasks requiring distributed problem solving methods may be envisaged, as e.g. inter-discipline planning or constraint satisfaction methods.

Therefore, this is the architecture adopted as basis for the interoperability approach developed in this thesis.

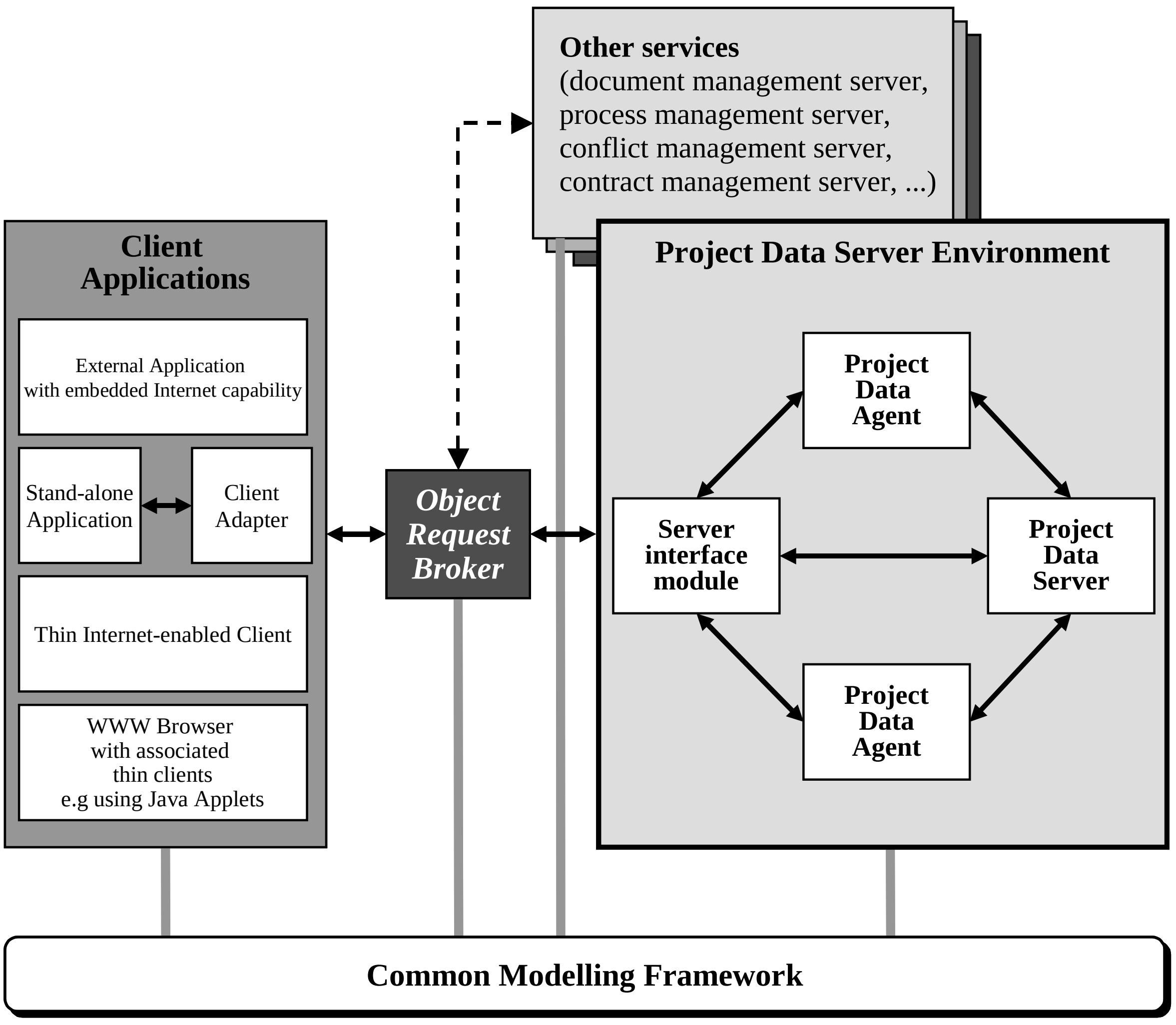

Other services
(document management server,
process management server,
conflict management server,
contract management server, ...)
Client Applications
External Application with embedded Internet capability
Stand-alone Application
Client Adapter
Thin Internet-enabled Client
WWW Browser with associated thin clients e.g using Java Applets
Object Request Broker
Project Data Server Environment
Project Data Agent
Server interface module
Project Data Server
Project Data Agent
Common Modelling Framework

Fig. 3.5: Principal architecture of the proposed client/server CEE system

3.6 System Interoperability

A highly flexible, distributed CEE system leads – due to the necessarily non harmonised models and heterogeneous software components – to a number of serious interoperability problems that need to be solved. In a broad analogy to the upper layers of the general ISO/OSI model of communication (cf. Rose 1989), they can be categorised as follows:

- **Systemic interoperability**, which has to deal with problems related to communication; (session/transport layer)
- **Semantic interoperability**, which has to deal with problems related to perception; (presentation layer)
- **Functional interoperability**, which has to deal with problems related to performance. (application layer)

In particular, systemic interoperability has to tackle the problems of the communication between a model-based project data server and an extensible set of heterogeneous applications, using different modelling paradigms and data structures that may not be known when the system is initially designed. This requires methods by which data is transported between the clients and the server, enabling the components of the CEE system to "**talk**" to each other.

Semantic interoperability has to tackle the problems of model data transformations between non harmonised models in a distributed model world. This requires methods by which the data can be adequately presented to each component of the system, enabling these components to "**understand**" each other.

At last, functional interoperability has to tackle the problems related to conflicting data resulting from concurrent user actions. This requires methods by which changes to the data can be appropriately applied, enabling the information processes to "**react**" to the changes in the material world.

As a whole, these three interoperability categories can be processed in a similar way as the ISO/OSI layers in a communication system, in the sequence "talk" – "understand" – "react". They present the main focus of this research and will be discussed in much more detail in the next five chapters[*)].

By this suggested *modularisation of the interoperability problems in CEE*, i.e. systemic – semantic – functional interoperability, a more comprehensive coverage of different data management tasks can be achieved, and separate concepts for the different types of problems can be developed. However, it is necessary to find methods to treat the individual interoperability tasks consistently, as part of a logically connected overall process. For this purpose, the use of a *high-level system-wide ontology* based on an initial idea from (Katranuschkov et al. 1997b) is proposed.

[*)] The concepts related to systemic interoperability are presented in chapter 4, and the concepts related to semantic interoperability are covered in chapters 5 to 7. Functional interoperability, which is not a conceptual but an implementation issue, is considered only partially in the scope of the work. However, the use of functional interoperability operations in the implemented prototype system is shown in the detailed example in chapter 8, and the formal definitions of these operations, as well as their relationships to model data states are presented in Appendix V.

An **ontology** is an explicit specification of a conceptualisation (Gruber 1993). The term is borrowed from philosophy, where an ontology is a systematic account of existence. In the CEE system, this means that what “exists” is that which can be represented.

The ontology of the CEE system is defined by a set of representational terms. These terms correspond to a well-defined subset of the objects of the modelling framework presented in section 3.4.

The ontology is used to describe the **ontological commitments** of the clients and the server in the CEE system, so that they can communicate about a mutual domain of discourse, the objects on the ontology level. A client **commits** to the ontology if its observable actions are consistent with the definitions in the ontology*).

Pragmatically, this means to define a common vocabulary with which queries and assertions can be exchanged in client/server interactions.

To be more explicit, let us consider what may happen when a client application requests information about a wall, for example to check code compliance for fire resistance. In an IFC-based modelling environment this information will be stored in the shared repository of the system as an *IfcWall* object and a number of related resource objects, such as *IfcMaterial*, *IfcPropertySet*, *IfcPoint* etc. An appropriate query to the project data server can easily be defined, but due to the different structuring and partially different semantics of a “wall” as understood by the server and the client, it is not so obvious as to what should be the server response.

One possibility is that the server “knows” the format and the content of the data needed by the client and translates the IFC objects accordingly. The drawback of this approach is that it would most probably lead to a dedicated server-side agent for each client application which would make application integration a complex and inefficient process.

Another possibility is that the server sends the IFC data in their original representation form and the application takes care to translate them to its internal data structure. In fact, this is how currently most known systems operate. However, this approach also leads to a *dedicated translator* for each application. The difference to the first approach is basically in shifting the implementation of the data transformations and the computational load from the server to the clients. A reduction of the number and the complexity of the required translators cannot be achieved.

A radical solution would be provided if each client commits to the full modelling framework which eliminates the need of any data transformation modules. Unfortunately, this solution has two problems. The first is the lack of comprehensive frameworks that can cover the information needs of a wide range of applications as needed in a general CEE system. The second is that even when such a framework is available, its schemas would normally be laid out for a broader scope, which makes them inefficient and verbose from application point of view. Therefore, in its pure form this approach is suitable only for systems with clearly defined, narrow scope and goals.

*) This idea is based on the *Knowledge-Level* perspective, introduced by Newell (1982) in the domain of distributed artificial intelligence. The knowledge level is a level of description of the knowledge of an agent that is independent of the symbol-level representation used internally by this agent.

The problems with the above approaches can be greatly overcome with the help of the suggested system-wide ontology. It provides for a more flexible, intermediate way for client/server interaction where the clients and the server have a common understanding of the high-level concepts of the environment to which they have committed, but may "use" these concepts differently at a lower level. The advantage of this approach is that common concepts can be more general and lean, encompassing just the necessary functionality to enable the discourse. It does not eliminate the need for appropriate translators - in fact, such translators are required now both at the server and at the client side. However, their realisation can be modularised and, consequently, considerably simplified because, different from the other described approaches, they can all (re-)use the common generic specifications provided by the ontology.

For the given example this would mean to define the concept of a wall in an unequivocal way which is independent of the specific (and probably different) data definitions of the server and the client, but provides sufficient means to execute operations on wall objects and to interpret the results of these operations properly.

How can this be accomplished?

Unfortunately, a commitment to a common high-level ontology guarantees only the consistency, but not the completeness of the discourse. Therefore, it is important to define the level at which the ontology is specified. This representation level is chosen on the basis of two criteria: (1) *lean content* (the client/server interaction should not be overburdened with too many concepts), and (2) *sufficiency* (the representation should be sufficient for unambiguous error-free discourse).

In accordance with that, in the proposed modelling approach for CEE the following components of the ontology level are suggested:*)

- Meta model
  (to provide a common explicit representation paradigm)
- Information Container
  (to provide a common method for packaging and transferring object data, regardless of their underlying model schemas)
- Generalised communication model
  (to provide a common communication mechanism, independent of the specific under-lying network protocols)
- Operations
  (to provide the basis for distributed object processing, as well as for mapping of remote methods to TCP/IP based requests/responses)
- Knowledge-based queries
  (to provide an advanced vocabulary for sophisticated agent-driven functions).

Thus, with the introduction of a system-wide ontology, a more specific and better "implementable" model of the information processes can be obtained, as shown on fig. 3.6.

*) These components and their inter-relationships are discussed in detail in chapter 4.

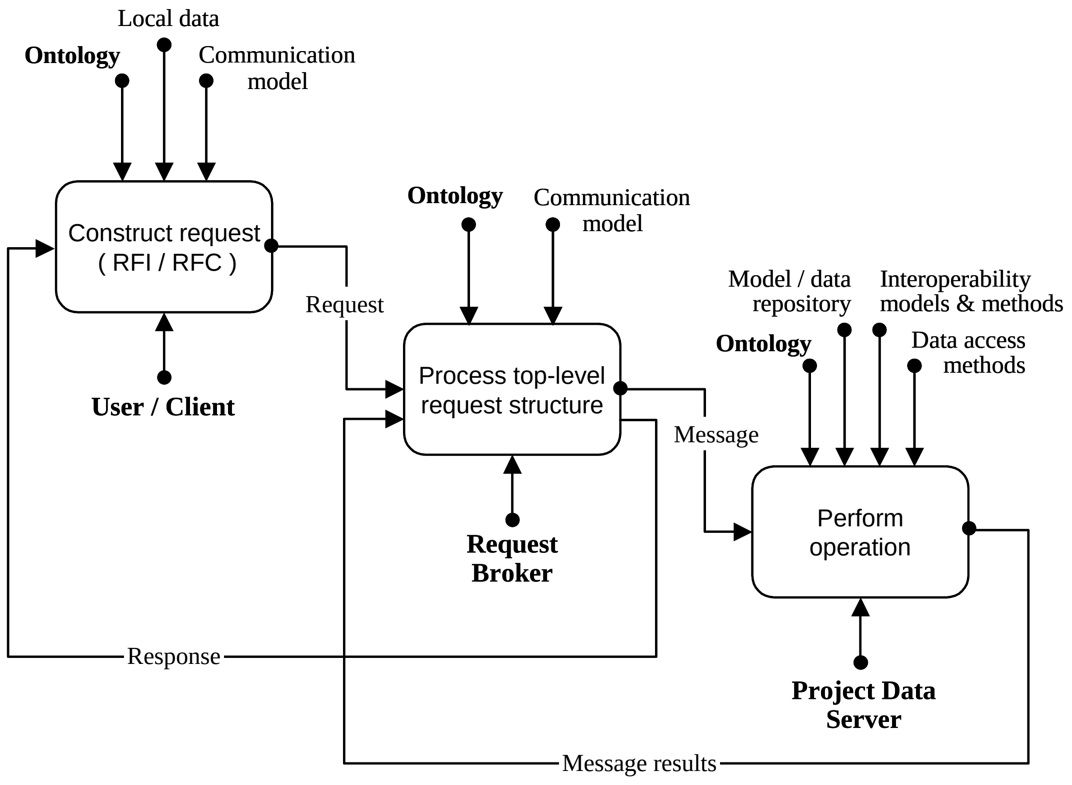


Fig. 3.6: Principal process model of client/server interaction in the CEE system in IDEF0

3.7 Concurrency

The last problem related to the suggested approach for the construction of a generalised conceptual framework for a distributed client/server CEE system, centered around a set of standardised conceptual data models, is the problem of *concurrency*.

In traditional integration approaches most often a shared project repository is proposed. However, in design work this may be counterproductive if the requirements for data consistency cannot be efficiently observed.

In a discussion about software design James Rumbaugh (1996) defines the “myth of the shared repository” in the following way:

> *“A popular approach to parallel development by multiple developers is the shared repository, in which each developer sees the latest version of each model element (classes and methods) in the entire project. Several development tools support this approach. This is bad for anything but a small project. Different developers get in each other’s way because their temporary changes are visible to everybody immediately. Each developer or development team needs a private workspace in which the developer can work with a stable version of the system until his or her changes can be ready to be shared with everybody.”*

In the above quotation, by substituting “classes” and “methods” by “slabs” and “beams”, “developer” by “designer”, and “software” by “building”, it is easy to see that much the same problems are relevant to building design as well.

In order to tackle these problems, a structuring of the concurrent design process by **discrete coordination points** is suggested, as illustrated on fig. 3.7. This means that instead of trying to guarantee a continuous consistency, coordination and reconciliation of diverging data is done by the designers themselves - “when needed” and “as needed”. This coordination process should be supported by appropriate change management tools. The consistency of the data is thereby not automatically guaranteed, but methods can be provided for user-driven monitoring and control. In this way, a high degree of concurrency of the “material” system, i.e. the real design environment is hoped to be achieved[*)].

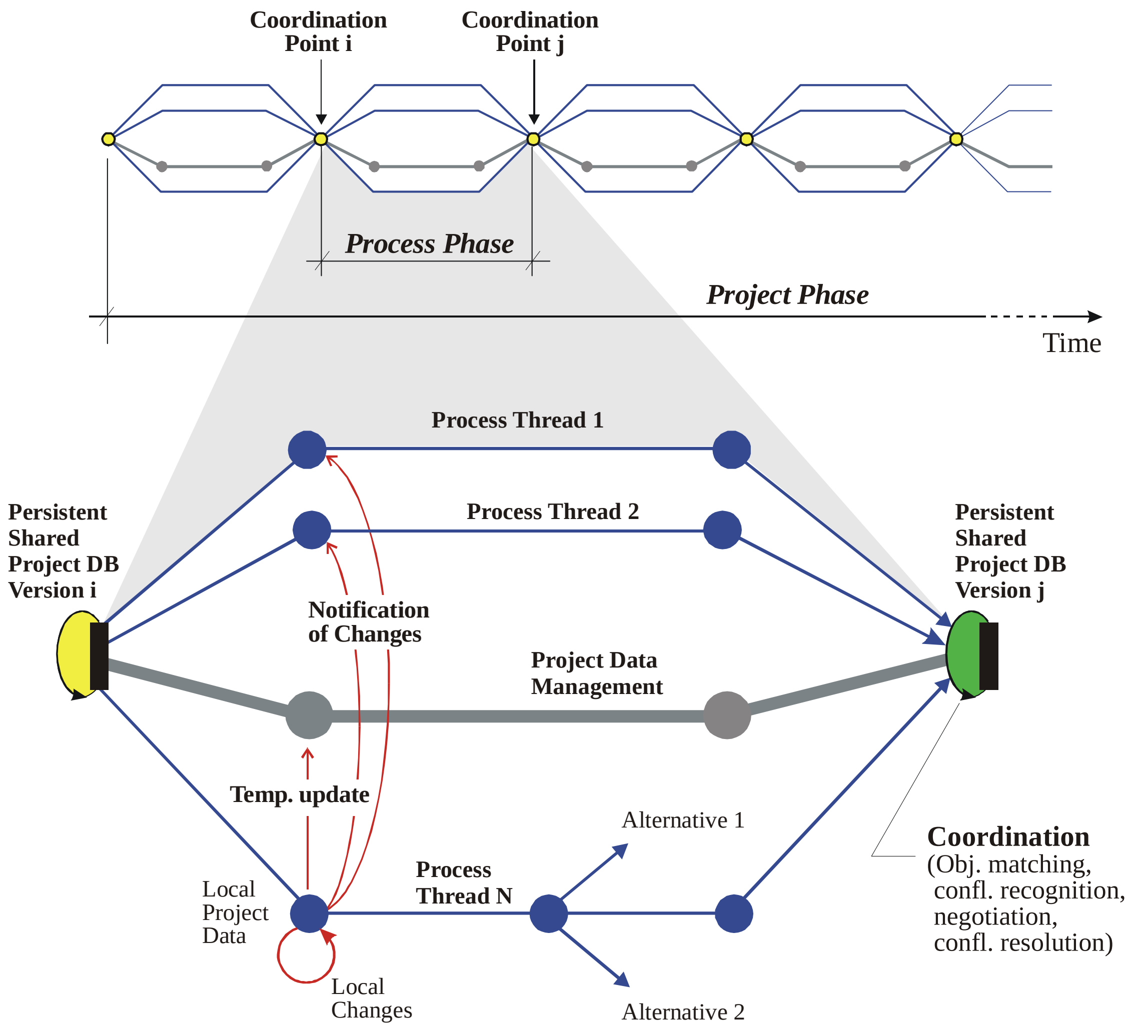


Fig. 3.7: Structuring of the concurrent project processes by explicit coordination points

[*)] The prototyped supporting tools for that kind of problems only provide some initial evidence of the validity of the suggested methods. Because data conflicts in complex data models can be of many different kinds, much research is still needed to determine the boundaries of the approach.

3.8 Discussion

Distributed software architectures have been intensively studied in the last years. There exist many successful applications for a large number of business cases in a wide range of society areas. The current trend is to install some kind of a client/server system on the World Wide Web to enable the creation of common virtual portals for cross-company collaborative work, facilitating document sharing, team communication and project-wide workflow management, as well as fast and secure information exchange. Less work has been done to provide product model based design environments, but projects like ToCEE, VEGA, RISESTEP and PIPPIN have developed convincing prototype solutions verifying the applicability of a model based approach.

By looking at such currently available developments, the choice of an Internet-enabled client/server system for CEE seems to lie on the hand. However, it would be superficial to assume this automatically as the only possible, or the most suitable solution. Besides, in order to determine the specific features of the system, available principal alternatives need first be examined. Therefore, many of the issues mentioned in the preceding sections had to be studied in detail before such decisions could be made. Three of these issues are revisited with a brief discussion below.

Enabling the collaborative work of distributed design teams

It is a common understanding that collaborative work requires an integrated data model and a common data repository (cf. West 1994; Hannus et al. 1995b; Wix 1996), but the need of a client/server architecture is less apparent. For example, if the objective is to support collaborative work in a single organisation at one physical location, a LAN solution providing faster access to the available computer resources may be preferred, and if cooperative decision making processes are of primary importance, agent-based architectures should also be considered. However, in my opinion, the client/server model provides the most general approach.

In contrast to a network solution, where the user is aware that there are several machines, and their location, storage replication, load balancing and functionality is not transparent, a client/server system appears as a single local machine, even when there are heterogeneous hardware and software components involved. This hides the complexity of the system from the end-users and allows them to concentrate on their real design tasks. It enables also more flexible software configuration because the possible heterogeneity of the components is abstracted by the underlying communication protocols.

These advantages of a distributed system to a networked system are manifested by agent-based systems as well. Moreover, the agent-based approach provides more powerful features because agents can act autonomously and proactively. Why, then, not an agent-based approach?

The reasons to prefer the simpler, “more conservative” client/server model are twofold. First, agent systems basically presume a well-defined closed model world. However, as already mentioned, such assumption cannot be made for a general-purpose environment where the component applications and their requirements cannot be predicted in advance[*)].

[*)] In fact, agent systems for non deterministic, dynamic environments have also been investigated in the area of distributed AI (cf. Russel & Norvig 1995). However, such systems are extremely complicated and are therefore hardly applicable for a practical CEE system in building design.

Second, in order to cooperate efficiently, agents do require more sophisticated architectures, computational methods and communication paradigms. By imposing such requirements on the clients of an integrated CEE system, the range of applications that can qualify for this approach would considerably be reduced, and valuable engineering software developed by traditional programming methods would not be possible to use.

In summary, the reasons to reject a pure agent-based approach are more of practical than of theoretical nature. However, as suggested in section 3.5, agent technology can also be applied in the frames of a client/server architecture provided that an adequate distribution of the roles and functions of the clients and the server is established.

Client/server roles and functions for efficient inter-discipline communication and information sharing

In principle, each distributed system includes three categories of functions: (1) presentation functions, (2) application functions, and (3) data management functions. Obviously, in a client/server system the presentation functions are most closely related to the end-user interfaces and must at least partially be allocated to the clients, whereas the data management functions, which should support the multi-user access to shared data, must be allocated to the server. So where is the dividing line?

There is no single answer to this question as illustrated on fig. 3.8 below.

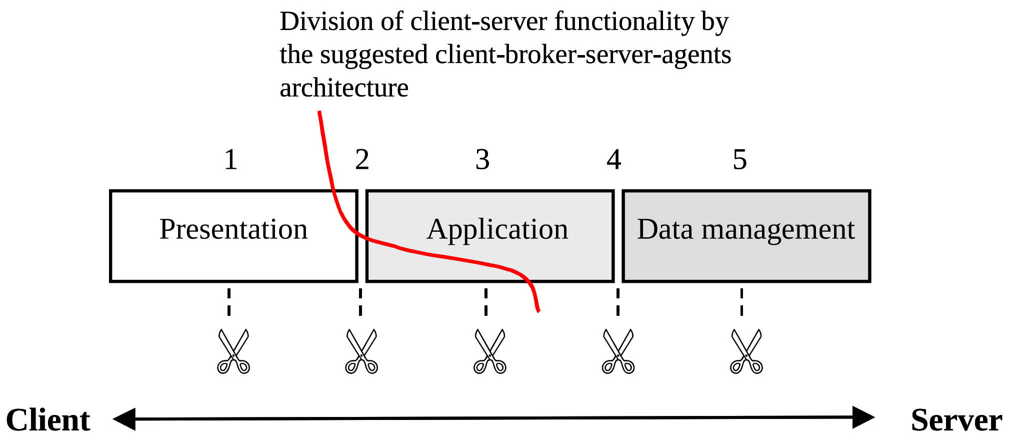

Division of client-server functionality by the suggested client-broker-server-agents architecture
1
2
3
4
5
Presentation
Application
Data management
Client
Server

Fig. 3.8 Client-server role distributions
(principal alternatives and suggested approach for the envisaged CEE system)

The first principal option, shown by cut 1 on the above figure, leads to a distributed presentation as provided by the X-Windows system. This is a useful basic technology, but it is not of any particular relevance to the needs of CEE.

The second option (cut 2) leads to a remote presentation system as provided e.g. by *Microsoft's Terminal Server*. It is also a basic technology, not directly relevant to the specific problems of CEE.

The third option (cut 3) represents a distributed processing paradigm which is characteristic for most agent-based approaches (cf. Müller 1993), as well as for all distributed object model architectures (cf. Harold 1997; Orfali et al. 1997). In contrast to the first two options, the specific interpretation of this kind of role distribution is of utmost importance to the overall functionality of the CEE system. Therefore, it will be separately discussed further below.

The fourth option (cut 4) represents remote database access solutions which are typically realised by using high-level languages such as SQL. This approach can easily be applied for STEP based product data as well, either on the basis of SDAI (ISO 10303-22 1998), or by implementing light-weight remote procedures to store/retrieve STEP physical files (ISO 10303-21 1994). In fact, this latter possibility has been successfully applied e.g. in the ATLAS and the COMBI projects. However, it provides very limited server functionality and is of almost no help for the tackling of the interoperability problems outlined in section 3.6.

The last option (cut 5) is again of little interest. Here the server plays merely the role of a file server providing no further functionality except explicitly requested file download/upload.

Thus, of all possible options for the distribution of the client-server roles in a CEE system, only the third option is effectively of interest. Its particular realisation strongly influences the overall design of the system and the methods by which interoperability can be achieved. A basic implementation approach is provided here by the three-tier architecture suggested by CORBA, which enables the distribution of application functionality by means of remote method invocation and a high-level interface definition language ensuring platform and programming language independence (cf. Orfali et al. 1997). However, it also restricts client-server communication to actively issued method calls by the clients, whereas the server takes only a passive role. This provides the needed functionality for many practical tasks, but is insufficient where interoperability and data consistency tasks are concerned. The tackling of such tasks requires an appropriate reaction of the server to changes in the data state, and not only in response to explicit client requests. For that purpose, the use of the agent paradigm as an extension to basic distributed object processing is more appropriate.

Beside the advantages mentioned in section 3.5, this architecture provides at least two more implementation related benefits:

1) It enables the separation of basic data management from advanced server functionality which makes the server realisation more modular and easier to extend;
2) It allows more flexible distribution of the client/server roles, as schematically shown on fig. 3.8, so that decisions of where and how to implement each particular function can be driven by actual technical demands and not by the basic software technology used.

However, what kind of agents are adequate to use? In (Russell & Norvig 1995) the following categories are identified: (1) simple reflex agents, acting according to situation-action rules, (2) reflex agents that keep track of the world by observing changes in the environment and continuously updating an internal data state, (3) goal-based agents which try to satisfy preset goals by means of search and planning algorithms, and (4) utility-based agents which evaluate the state of the environment to weigh up the likelihood for a successful action against the importance of its goals. A more general classification defines the first two categories as reactive agents, and the second two as deliberative agents (cf. Müller 1993). For certain specific tasks any of these agent types may be most appropriate, but the typical case for the client-server model is that of a *reflex agent with internal state* which represents a part of the system that performs information preparation and exchange on behalf of a client or on behalf of the server (cf. Orfali et al. 1999). In my opinion, this approach fits best into an object-oriented PDT environment. Therefore, it has been adopted for the server prototype developed in this thesis. The suggested method to address such agent functionality is discussed in section 4.7 of chapter 4.

Adequacy of the IFC models for CEE

In section 2.3.2 it was mentioned that the IFC models seem to be adequate for the envisaged CEE system as they closely follow the decomposition approach providing the needed modularisation of the CEE data structures. This is also schematically shown on fig. 3.3 in section 3.4. However, this figure shows also that IFC was not expected to provide all information needed for CEE. Therefore, before taking a final decision to use IFC as basic reference model, a more detailed examination of the IFC framework had to be carried out. For this purpose, the principal decomposition matrix for CEEMF proposed in section 3.3 has been used as baseline. The results of the undertaken study are summarised in table 3.3 below.

Table 3.3: Adequacy of the IFC models for CEE

Information type	Information subtype	Provided by IFC
Construction information	Product information	Yes
	Process / activity information	Partially
	Documents	Meta data partially provided
	Legal data	No
CEE system information	Product information	No
	Process / activity information	Partially
	Legal data	No
Generic information	Meta information	No
	Object model	implicit (EXPRESS)

The performed qualitative evaluation showed that the IFC models are indeed well suited for a prototype environment for validation purposes, especially as far as the overall structure and construction product information are concerned. Due to the currently weak harmonisation w.r.t. software applications they provide also a good test-bed for the model mapping approach proposed in this thesis.

However, there are also many gaps that need to be filled before an IFC based CEE system can be used in practice. Of these, most strongly missed is the representation of CEE system information. This aspect is only partially seen in the scope of IFC development, but it is of utmost importance both for the conceptual design and for the implementation of an interoperable CEE system. The next chapter presents a novel approach to fill some of these gaps.

Chapter 4: Systemic Interoperability

You see, wire telegraph is a kind of a very, very long cat. You pull his tail in New York and his head is meowing in Los Angeles. Do you understand this? And radio operates exactly the same way: you send signals here, they receive them there. The only difference is that there is no cat.

– Albert Einstein (in an interview, when asked to describe radio)

Systemic interoperability presents the first category of interoperability problems that need to be tackled in a concurrent engineering environment. It is the basis upon which all other methods are built.

In this chapter a formal approach for the solution of the main systemic interoperability problem is proposed, i.e. the tackling of the communication between a model-based project data server and an extensible set of heterogeneous applications, using different modelling paradigms and data structures that may not be known when the system is initially designed. The developed approach provides a methodology for the representation and exchange of all types of information supported by the system, and facilitates the use of object-oriented and knowledge-based functionality in a unified, platform-independent way.

The structure of the presentation follows the logic of the suggested layered set of upward-linked specifications.

At first, following a brief introduction to the basic ideas of the approach, the concept of *Information Containers* is detailed. It provides a simple method for representing any of the data types that may be dealt with in CEE. On its basis, the ontological commitment of all components of the system is achieved.

Second, the proposed generalised *Communication Model* is presented. It utilises the Information Container data structures to provide a specification of client/server transactions on a high level of abstraction. This allows uniform coverage of many basic communication techniques, such as TCP/IP sockets, remote procedure calls, URL-based queries and so on.

Third, the concept of language independent *object-oriented project data operations* is introduced, and a set of functions that can be used as “building blocks” for the provision of more advanced project data services is identified. The discussion includes principal suggestions for the proper maintenance of the model data states within and between client/server sessions, in accordance with the concurrent engineering requirements identified in the previous chapter.

As a further logical step, a formalism for the representation of *knowledge-based queries and assertions* is defined, and a technique for packaging them in Information Containers is presented. The developed specification enables the access to advanced knowledge-based functions of the project data server by all kinds of client applications. Some of the capabilities of this approach are demonstrated by a number of illustrative examples.

At the end of the chapter, the scope and the applicability of the developed concepts are briefly discussed, and envisioned future extensions are outlined.

4.1 Basic Concepts

In short, systemic interoperability can be defined as the ability of the system components to work together in a coherent way for the solution of complex tasks that cannot be solved individually. This "ability" requires more than simple "tell and ask" sequences using a prescribed presentation format.

Indeed, in the envisaged client/server system, comprised of a central project data server and a number of heterogeneous applications, a primary concern obviously is the establishing of adequate communication methods to link these applications with the server. From the standpoint of application developers and end-users of the CEE system, this kind of interoperability can be seen simply as a problem of "asking questions" and "receiving answers", similar to telephone connection. However, the requirement for concurrent access to shared data raises more serious systemic interoperability problems at the server side, which cannot be solved merely by appropriately chosen and implemented communication techniques. Most of these problems are in fact a matter of appropriate *conceptualisation.*

In general, a client/server architecture is a simple software model based on the following three design principles (cf. Tanenbaum 1988; Jamsa et al. 1996):

1) Roles and functions are asymmetric, i.e. the initiative to start an interaction is always at the client side, whereas the server is responsible to listen to client requests and perform the requested services.
2) The topology of the system is N:1 (N clients, 1 server), with the server acting as a static component, and the clients as dynamic components of the environment that cannot be configured in advance. Consequently, the services a server can perform and the protocols used to talk with it must always be known in advance, i.e. the clients adapt to the server, and not the server to the needs of the clients.
3) Client/server interaction always consists of the three steps Request-Perform-Respond, as shown on fig. 4.1.

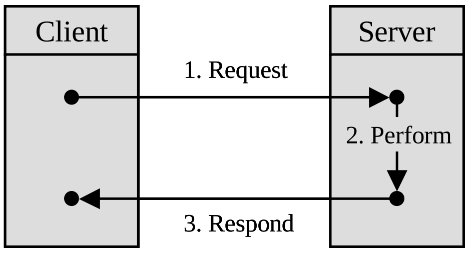


Fig. 4.1: Principal client/server interaction

However, as illustrated on the next fig. 4.2, in practice this does not work as simple as that.

First, the CEE system may have to deal with very different applications, for which different communication techniques may be suitable. A highly interactive application may prefer an interactive communication method, where the user is enabled to "talk" to the server directly, e.g. by means of a URL-based connection utilising the HTTP protocol; an analysis program may need to access the server data without any user interaction, which makes FTP or direct socket communication more appropriate; an advanced knowledge-based application may want to trigger sophisticated queries and integrate the results directly in its data structures, which makes the use of a distributed object model paradigm the appropriate choice, and so on.

This results in a requirement to support different communication paradigms which is not at all typical for standard client/server implementations.

Secondly, according to the requirement for concurrent access, simultaneous requests for changing the data of one and the same object have to be tackled. Even though it was suggested that the model data of the individual users should be kept and processed separate from each other, they are nevertheless representations of the same physical objects and have to be maintained consistent. This results in a requirement for data consistency as by distributed DBMS realised on the basis of the client/server paradigm (cf. Dadam 1996).

Thirdly, in engineering design a logically non separable unit of work normally requires a great amount of information for a long period of time (hours, days, or even weeks). When this information is retrieved from the server, it will normally not remain unchanged. However, during the same time it may also be needed (and modified) by others. Unfortunately, the usual transaction mechanisms provided by database systems are inappropriate for this normal design procedure[*)], and the standard implementation of "write locks" is even counterproductive[**)] (Rumbaugh 1996). Thus, it is necessary to support *long transactions*, check-in/check-out methods and versions to enable concurrent update operations over long periods of time (cf. Encarnação & Lockemann 1990; Herrmann 1991).

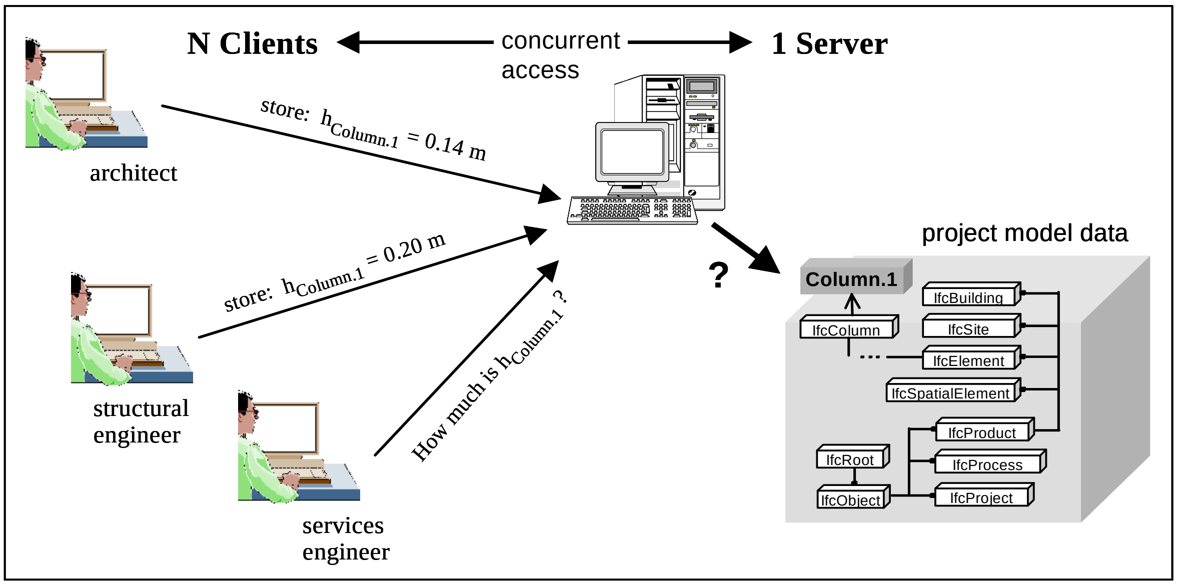


Fig. 4.2: The general systemic interoperability problem on the example of requests for the data of the object "Column.1"

In addition to the above, sophisticated engineering applications may need to cooperate with the server actively, by requesting complex sets of data that can only be provided by advanced server-side functionality. There are at least three reasons for this requirement:

*) Standard transaction mechanisms used in business DBMS are not satisfactory for engineering design work because (1) they are normally transient and may be lost e.g. by system crashes, and (2) encountered consistency conflicts automatically abort a transaction, instead of creating a new version of the conflicting data in order to try to recover consistency later.

**) A normal write lock inhibits any further update operations on the locked data until the application that locked them removes the lock. Hence, a designer, e.g. the structural engineer, may have to wait for days to get access to the data of another designer, e.g. the architect. This will result in a largely sequential design process, far from the objectives of CEE.

- reduction of communication bottlenecks (instead of constantly querying individual properties of the data, the application can receive the full information it needs in a single transaction),
- providing extended features of the model data that are not explicitly represented in the model data structures, but can be deduced by appropriate procedures (making the models “intelligent participants” in the design process), and
- utilising the local access of the server to all project data, which may be difficult for or even forbidden to the application.

In order to satisfy these requirements, the design of an interoperable CEE system must incorporate:

1) an **appropriate software architecture**, which is adaptable to different communication techniques and different project configurations, with different users and applications;
2) **description methods** enabling the uniform treatment of all possible types of client/server interactions;
3) an appropriate **model of the project data repository** enabling the maintenance of its consistency, but not restrictive w.r.t. the concurrency in the design team work;
4) **implementation methods** providing the use of the developed concepts in the running system.

The proposed systemic interoperability approach detailed in the following sections combines all these design aspects in a coherent way. Its key ideas include:

1) The development of a consistent high-level representation of the data transferred between the client applications and the server on the basis of the principle of a system-wide ontological commitment introduced in the previous chapter (this issue is provided by the Information Container specification presented in section 4.2, and the generalised communication model presented in section 4.3);
2) The development of a formal method for the definition and implementation of object-oriented operations, enabling the use of different communication paradigms, and providing controlled access to the data repository of the project data server (this issue is covered in sections 4.4 and 4.5, where the principal approach for the definition and implementation of operations is detailed, and a set of basic data management operations is suggested);
3) The alignment of the proposed types of operations with the management of the data in the project data repository, including the tackling of access rights, data states, availability of the operations w.r.t. data states etc. (this issue is discussed in section 4.6);
4) Extending the basic object-oriented functionality provided by the introduced concept of operations by a formalism to specify knowledge-based expressions, and an appropriate representation of the data models at the project data server, which is capable to accommodate knowledge-based functionality (the suggested formalism for the use of knowledge-based search expressions embedded in object-oriented operations is detailed in section 4.7, and the knowledge representation approach applied for the project data server is discussed in chapter 8, after the introduction of all conceptual issues related to the environment);
5) Alignment of all component specifications in a layered, coherent representation.

4.2 Information Container

The “Information Container” concept introduces the most fundamental data representation types that can be consistently used on the ontology level of the proposed environment. Its goal is to enable the packaging of *any* supported data definitions by a *uniform representation formalism*, such that the specification of a generalised client/server model of communication, and the coherent definition of more specific project data operations can be achieved. The objectives of that formalism are defined as follows:

1) Provide a neutral form for compact representation of the data contained in EXPRESS models, including both full and partial object instantiations;
2) Enable the exchange of individual product data and of models as a whole, including the use of different data exchange formats, such as STEP physical files, VRML, XML etc.;
3) Develop a structured representation, enabling the implementation of object-oriented APIs for use by applications written in object-oriented languages;
4) Provide a possibility for the embedding of expressions, enabling the use of knowledge-based server functionality and the construction of complex queries;
5) Define a syntax enabling the externalisation of all data structures to support stream-based information exchange with applications that are not using the object-oriented paradigm;
6) Ensure flexibility and extensibility of the specification.

As the proposed environment is based on EXPRESS, a natural idea to approach the problem would be to use the SDAI dictionary schema specified in STEP (ISO 10303-22 1998). Indeed, this schema can provide a useful basis, as it defines a data dictionary, capturing the meta information about the data instances in models contained in data stores. SDAI implementations typically utilise this schema in order to provide the necessary flexibility and the independence of the data access operations from the particular data models in use. Details of this approach are discussed e.g. in (Loffredo 1998). However, taken alone, the SDAI dictionary schema is not sufficient to provide the functionality suggested by the above objectives as it does not address several required features, such as concurrent access by multiple applications, access to remote data, global model identification, long transactions. Besides, SDAI is mostly used for *working form*, i.e. in-memory implementation of STEP APs. Research on its possible use in database or knowledge-based environments is up to now limited to a very few efforts (cf. Krebs & Lührsen 1995; Loffredo 1998). Therefore, a reduced, but sufficient adaptation of the data dictionary schema is applied. It provides a consistent basis for the overlaid specifications that will be introduced in later sections.

In this section, the structure, the components and the formal syntax of the **Information Container externalisation** are presented.

The details of the prototyped *Java language binding* for the proposed **Information Container API**, tested in the developed sample clients, is provided in Appendix II.

4.2.1 Structure and components of the Information Container specification

The Information Container specification is comprised of a small number of basic data types, structured as shown on fig. 4.3 below. EXPRESS-G is used to emphasise the strong relation to EXPRESS data types, as well as to maintain the uniformity of all specifications. However, the Information Container data types are *not* elements of an EXPRESS data model in the usual sense. Rather, they provide a method for the explicit formulation of meta model concepts, enabling the development of a variety of services for various purposes.

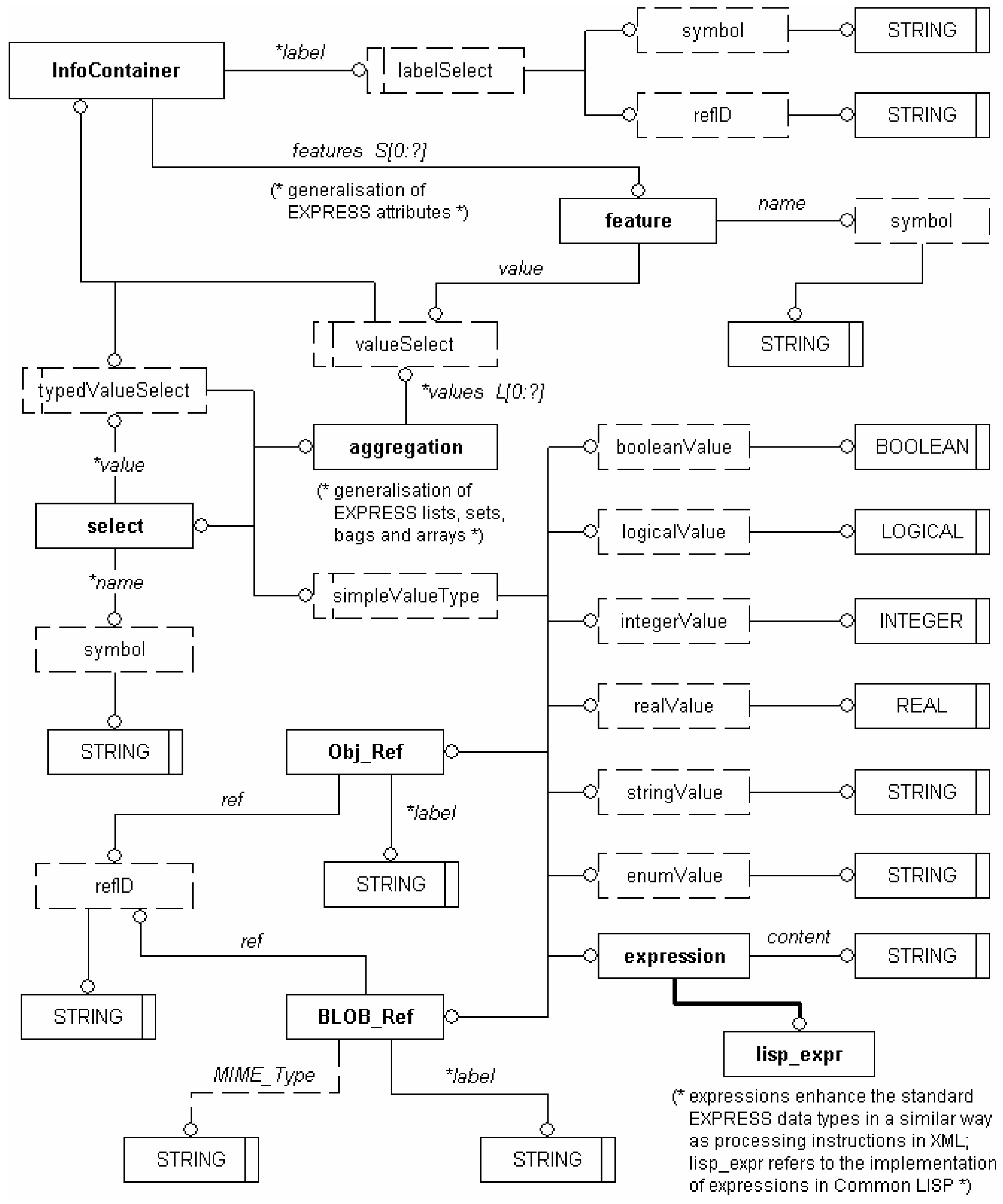


Fig. 4.3: Overview of the Information Container data structures in EXPRESS-G

The main element of the representation is the ***Information Container*** object itself. Each of its instances is addressable by a unique *label*, which can be a symbol or a reference to any type of data, supported in the environment (a high-level concept, an object class, a specific object in a model, or even that model itself).

The content of an Information Container is provided by a set of ***features***. Each such feature is represented by a name-value pair, where the name is a symbol, and the value can be of any of the data types defined in the Information Container schema, including *Information Container*, but excluding *feature*[*)]. This allows the reference of object attributes by name, providing an option for the representation of incompletely instantiated objects, and even for the definition of ad hoc properties of objects.

The value of a feature can be of one of the following categories: (1) simple data type, (2) structured data type, or (3) Information Container.

A ***simple data type*** cannot be further decomposed. This category includes the basic EXPRESS data types, along with *object references*, *references to binary large objects* (*BLOB*s) and *expressions*. In addition, the type **`symbol`** is defined in order to distinguish between strings and names, such as class/attribute names and enumeration items. This is not only conceptually preferable, but enables also the use of symbolic processing.

An *object reference* (**`Obj_Ref`**) provides the mechanism to address remote data objects, or to execute remote operations on remote data objects.

A *BLOB reference* (**`BLOB_Ref`**) provides the mechanism to process data files by Information Container based operations. By associating BLOBs with upload/download operations the transfer of exchange files can be achieved in a transparent way.

An *expression* provides the possibility to embed processing instructions in Information Container objects, enabling the specification of more sophisticated server operations[**)].

The ***structured data*** types include **`aggregation`** and **`select`**.

An **`aggregation`** is a set of data of one and the same type. This can be any data type defined in the schema, including Information Containers and aggregations. Thus, an aggregation can be used to construct multidimensional vectors of any type. The requirement for uniformity of the underlying element type is not a conceptual restriction, but a language feature.

Finally, a **`select`** is identical in meaning to the EXPRESS select type and is introduced for the same purpose.

[*)] The rationale for excluding features is: (1) to provide for a simpler, easier to parse syntax, and (2) to reduce the complexity of the model by eliminating the possibility to define "attributes of attributes", which is possible by some advanced representations, e.g. frames, but is not provided in the classical object-oriented approach.

[**)] The prototype implementation of the Information Container schema developed in this thesis allows only the use of Common LISP expressions (cf. Steele 1990). This provides a simple method to define more complex queries, but limits cross-platform applicability to implementations written in Common LISP. Therefore, *lisp expressions* are not supported bi-directionally in the communication model presented in section 4.3. Thus, whilst a client can submit requests containing lisp expressions, which will be "understood" by the project data server as its implementation is based on Common LISP, the use of lisp expressions in responses is not possible.

4.2.2 Formal syntax specification of the Information Container externalisation

The technical specification of the syntax for Information Container externalisation is comprised of three parts: (1) *tokens*, providing low-level definitions for the allowed lexical elements, (2) *grammar rules* presenting all high-level structures, and (3) *informal propositions*, including additional information not covered in the formal specifications for conciseness, or because it presents imported constructs described in other literature sources. From hierarchical viewpoint, corresponding to the schema provided on fig. 4.3 above, the primary rule of the Information Container syntax is rule (28).

Tokens

The following productions define the tokens used in the Information Container externalisation. Except where explicitly stated, no `whitespace` characters are allowed within the text matched by a single syntax rule for a token.

Character classes:

```
(1)  whitespace   = ASCII-SP | ASCII-HT | ASCII-CR | ASCII-LF |
                    ASCII-FF .
(2)  letter       = 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' |
                    'h'.| 'i' | 'j' | 'k' | 'l' | 'm' | 'n' |
                    'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' |
                    'v' | 'w' | 'x' | 'y' | 'z' |
                    'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' |
                    'H'.| 'I' | 'J' | 'K' | 'L' | 'M' | 'N' |
                    'O' | 'P' | 'Q' | 'R' | 'S' | 'T' | 'U' |
                    'V' | 'W' | 'X' | 'Y' | 'Z'.
(3)  digit        = '0' | '1' | '2' | '3' | '4' | '5' | '6' |
                    '7' | '8' | '9' .
(4)  special_char = '!' | '”' | '$' | '%' | '&' | '.' | ',' |
                    '*' | '+' | '-' | '(' | ')' | '[' | ']' |
                    '{' | '}' | '<' | '>' | '=' | '@' | '?' |
                    '#' | '^' | ':' | ';' | '’' | '`' | '~' |
                    '|' | '/' | '_' .
(5)  escape_char  = '\' .
(6)  escape_seq   = '\n' | '\r' | '\t' | '\\' .
(7)  quote_char   = '\”' .
(8)  non_q_char   = letter | digit | special_char | escape_seq .
```

Keywords:

```
(9)  TRUE    = 'true'.           (12) BLOB    = 'BLOB'.
(10) FALSE   = 'false'.          (13) OREF    = 'oRef'.
(11) UNKNOWN = 'unknown'.        (14) LISP    = 'LISP'.
```

Lexical elements:

```
(15) boolean = TRUE | FALSE .
(16) logical = TRUE | FALSE | UNKNOWN .
(17) longint = [ sign ] { digit }+ .
(18) real    = [ sign ] { digit }+ '.' { digit }*
               [ { 'E' | 'e' } [ sign ] { digit }+ ] .
```

```
(19) string  = '"' { non_q_char | quote_char | ' ' }* '"' .
(20) symbol  = letter { letter | digit | '_' }* .
(21) keyword = [ '_' ] symbol .
(22) refID   = modelID | objID .
(23) modelID = ID .
(24) objID   = [ modelID ':' ] ID .
(25) ID      = symbol [ '.' { symbol | longint } [ version ] ] .
(26) version = ';' longint .
(27) sign    = '+' | '-' .
```

Grammar rules

The following high-level production rules specify how the tokens can be combined into valid Information Container constructs. To avoid ambiguity, where necessary `whitespace` characters may be used as separators between the individual tokens.

```
(28) InfoContainer       = label '(' { feature }* ')' .
(29) label               = symbol | refID .
(30) feature             = symbol ':' valueSelect .
(31) valueSelect         = typedValueSelect | InfoContainer .
(32) typedValueSelect    = simpleValueType | structuredValueType .
(33) simpleValueType     = literal | Obj_Ref | BLOB_Ref | expression .
(34) structuredValueType = aggregation | select .
(35) aggregation         = '[' { valueSelect }* ']' .
(36) select              = symbol '(' selectValueType ')' .
(37) selectValueType     = literal | Obj_Ref | structuredValueType .
(38) literal             = boolean | logical | longint | real |
                           string | symbol .
(39) Obj_Ref             = OREF '(' refID ')' .
(40) BLOB_Ref            = BLOB '(' refID ')' .
(41) expression          = lisp_expr .
(42) lisp_expr           = '(' LISP lisp_form ')' .
```

Informal propositions

1) In accordance with the given syntax specification a straight-forward **mapping of entities defined in EXPRESS data models to Information Container elements** is possible, as follows:
   - class names and attribute names are represented as symbols;
   - enumeration items are also represented as symbols - this does not contradict to the EXPRESS syntax because class names and enumeration items are not allowed to have identical names within the same data model;
   - user-defined data types are represented with their underlying base types (similar to the ISO 10303-21 format);
   - `SELECT` types are always prefixed with the name of the selected element, e.g. `IfcPositiveLengthMeasure`(3.5);
   - lists, sets, bags and arrays are represented through aggregation structures;
   - references to object instances are represented by using the `OREF` construct;

- `INTEGER` is represented as `longint`, and `NUMBER` is represented according to the respective actual value (`longint` or `real`);
- `BOOLEAN`, `LOGICAL`, `REAL` and `STRING` are represented according to rules (15), (16), (18), and (19).

2) **`longint`** values may contain as many digits as permitted in the internal representation of long integer numbers, normally in the range `[ -2`^{64} `; 2`^{64}`–1 ]`.

3) **`real`** values should be interpreted as double precision floating point numbers in any specific Information Container implementation.

4) ***lisp_form***, as introduced in rule (42), is an external construct intended as a substitute for any special rules for the definition of arithmetic or relational expressions. A *lisp_form* can be a valid self-contained Common LISP top-level form or a lambda expression as defined in (Steele 1990), but should always return a single Information Container construct. Only local variable bindings are allowed in a *lisp_form* used as `simpleValueType` in an Information Container construct.

5) An internal representation for each **keyword** in the form **`'_symbol'`**, as specified in rule (21), is provided for the cases where a reserved word from the Information Container specification exists also as a name of an element in an EXPRESS schema and thus cannot be resolved unambiguously by the Information Container Parser. For example, if the name “`BLOB`“ exists in a data model, **`'_BLOB'`** instead of **`'BLOB'`** should be used to refer to a `BLOB` element in an Information Container.

6) Finally, although the Information Container syntax itself does not impose explicitly any case-sensitivity rules, **upper and lower case** in all names defined in EXPRESS schemas should be preserved because an implementation written in a language like C++ or Java may rely on such case-sensitive names.

4.2.3 Information Container examples

Example 1: *adapted from (ISO 10303-42 1994)*

Given the specification of the following geometric entities:

```
ENTITY line;
  pnt : cartesian_point;
  dir : vector;
END_ENTITY;
ENTITY vector;
  orientation : direction;
  magnitude   : length_measure;
END_ENTITY;
ENTITY direction;
  direction_ratios : LIST [2:3] OF REAL;
END_ENTITY;
ENTITY cartesian_point;
  coordinates : LIST [1:3] OF length_measure;
END_ENTITY;
TYPE length_measure = REAL;
END_TYPE;
```

the `line` instance **`L1`**, defined by **`L1`**`(u) =` **`P1`** `+ u` **`V1`**`,`
where **`P1`** `= < 0.0 0.0 0.0 >,` **`V1`** `= < 1.0 1.0 0.0 >` and `0 ≤ u ≤ 2.5,`
can be represented by the following four Information Container:

```
1. line.L1(pnt:oRef(P1) dir:oRef(V1))
2. cartesian_point.P1(coordinates:[0.0 0.0 0.0])
3. vector.V1(orientation:oRef(D1) magnitude:2.5)
4. direction.D1(direction_ratios:[1.0 1.0 0.0])
```

Comment:

This simple example illustrates how different types of attributes (real numbers, lists, object references) can be mapped to Information Container constructs. For comparison, the same example data would have the following representation in STEP physical file format (ISO 10303-21 1994):

```
#1 = LINE(#2,#3);
#2 = CARTESIAN_POINT((0.0,0.0,0.0));
#3 = VECTOR(#4,2.5);
#4 = DIRECTION((1.0,1.0,0.0));
```

The key differences between these two representations are as follows:

1) In the Information Container specification attributes are referenced by name, whereas in ISO 10303-21 they are positional. This allows the representation of evolving data which is not possible with the STEP physical file format.

2) The object class is not included in the Information Container representation. It can be deduced from the assumed rules for constructing object identifiers, and is also explicitly represented in a client request, as detailed in section 4.3 below.

3) On the opposite, with Information Containers the system-wide object identification does *always* participate in an exchange structure, whereas in the STEP physical file format objects are only identifiable in the scope of the exchange file. This particular feature is of utmost importance for maintaining the data integrity in the total system. In a STEP physical file references to other objects are interpreted as pointers to the respective lines in the file where these objects are specified. In contrast, an **`oRef`** in an Information Container is a reference to the respective actual object in the model itself, i.e. an **`oRef`** does connect Information Container meta data with the real object data in the data store.

4) A STEP physical file represents the whole model data, or at least a self-contained portion of this data with no unresolved pointers to external objects, whereas the four Information Containers shown above are in fact four *independent* data structures. The sample object reference values and the respective labels of the Information Containers (e.g. **`oRef(V1)`** and **`vector.V1`**) are given similar names only for the purpose of a more explicit illustration of the links between the objects in the EXPRESS schema. However, the actual links are provided through the actual object instances, and not through the Information Container labels. The option of using an object reference as a label is important for the object-oriented realisation of the Information Container API presented in Appendix II, and not for the Information Container externalisation.

Example 2: *adapted from IFC 2.0 (IAI 1999c)*

“Beam” objects are defined in IFC 2.0 as follows (arrows indicate the inheritance path down to *IfcBeam*):

```
ENTITY IfcRoot;
  GlobalID : IfcGloballyUniqueID;
  OwnerHistory : IfcOwnerHistory;
  Label : OPTIONAL STRING;
END_ENTITY;
ENTITY IfcObject SUBTYPE OF (IfcRoot);
  UserDefinedType : OPTIONAL STRING;
  DocumentReferences : SET [0:?] OF IfcDocumentReference;
  ...
END_ENTITY;
ENTITY IfcProduct SUBTYPE OF (IfcObject);
  LocalPlacement : IfcLocalPlacement;
  Representations : SET [0:2] OF IfcProductRepresentation;
  ...
END_ENTITY;
ENTITY IfcElement SUBTYPE OF (IfcProduct);
  ...
END_ENTITY;
ENTITY IfcBuildingElement SUBTYPE OF (IfcElement);
  HasMaterial : OPTIONAL IfcMaterialSelect;
  ...
END_ENTITY;
ENTITY IfcBeam SUBTYPE OF (IfcBuildingElement);
  calcBeamSectionArea : OPTIONAL IfcAreaMeasure;
  calcBeamVolume      : OPTIONAL IfcVolumeMeasure;
  ...
END_ENTITY;
TYPE IfcGloballyUniqueId = STRING;
END_TYPE;
```

According to this specification, an entity instance of **`IfcBeam`** may e.g. have the following representation as an Information Container:

```
IfcBeam.12345 (
  GlobalID:"12345"
  OwnerHistory:oRef(IfcOwnerHistory.1175223)
  LocalPlacement:oRef(IfcLocalPlacement.9731211)
  calcBeamSectionArea:1200.0
  Label:"BEAM-1" )
```

Comment:

From this example, using a simplified form of the IFC definition for *beam* objects, it is evident that only attributes that have actual values are contained in an Information Container for a given object instance. Thus, it is possible e.g. to exchange the information about a *beam* in an early design phase where it may already have a defined location and cross section area, but other properties are not yet available. Moreover, the Information Container for the **`IfcBeam.12345`** instance contains only the top-level structure defined in the **`IfcBeam`** class. References to related objects, if existing, are provided as **`oRef`**s and can, but must not be queried by the client application. In this way, the transmission of data along a communication channel is reduced to the minimal necessary content, which contributes to the avoidance of communication bottlenecks.

Because the attribute representation is *not* positional, their order within the Information Container is unimportant. In the object model of the Information Container API for Java[*)], a set of methods are provided which enable the easy access and processing of all its components by client applications.

4.3 Generalised Client-Server Communication Model

The generalised client-server communication model presents the second fundamental information representation concept in the proposed approach. Its objectives are as follows:

1) Provide a high-level abstraction of client/server communication, enabling the identification of hardware (host server, clients), software and users, and the construction of sessions in which authentication, access rights and transactions can be properly managed;
2) Provide a conceptual specification of request and responses, enabling the coverage of a wide range of communication paradigms used on the Internet, such as socket connection, remote procedure calls, remote method invocation à la CORBA or Java RMI, and HTTP-based CGI-scripts (cf. Jamsa et al. 1996);
3) Enable the use of Information Container data structures in request/response messages, and the linking of this data with the model data of the server and the client applications;
4) Enable synchronous/asynchronous execution of short and long transactions.

This section presents the formal specification of a proposed communication model schema satisfying the above objectives, as well as two illustrative examples of its suggested use. Further considerations w.r.t. the relationship of the developed communication objects to the operations on the model data are provided in sections 4.4 - 4.6.

4.3.1 Structure and components of the communication model

The principal structure of the communication model is shown on fig. 4.4 below.

The main component of the schema is the abstract entity ***CommunicationEvent***, with its subclasses *Request* and *Response*. Each *Request* object is associated with a *CEEsession*, and each *Response* is linked with its corresponding *Request*.

A ***CEEsession*** object may itself be associated with several *CommunicationEvent*s, which allows the grouping of client-server communication in sessions, overcoming the limitations of stateless protocols, such as HTTP. It contains information about the user, the client application, the host computer running the application, and the specific communication methods supported by the application. The external resources referenced by *CEEsession*

[*)] see appendix II.

are taken from the IFC 2.0 project model specification (IAI 1999c) for the purpose of better harmonisation with the IFC Project Model. This does not imply any design compromise as both *IfcApplication* and *IfcActorSelect*, enabling the access to *IfcPerson*, *IfcOrganization*, *IfcPersonAndOrganization* and *IfcActorRole* objects, provide adequate specifications for the derivation of the actual user access rights needed in the CEE system.

CommunicationEvent objects, i.e. ***Request*s and *Response*s**, are created by the Request Broker on the basis of the Communication Model schema. Their externalisation is provided by representing them as Information Containers, and their object-oriented use is supported by the object classes of the Information Container API, and the capability for remote method invocation provided by the generic *execMeth* method of the *ObjectRef* class (see Appendix II).

The inherited top-level attributes of *Request*s and *Response*s present the necessary abstraction enabling the definition of operations on the ontology level of the framework. The attribute *concept* allows to reference meta object information explicitly, and not only by implicitly imposed programming rules, *OID* provides the link to the object that is responsible for the execution of the requested operation, and *meth* provides the name of the operation itself. The optional attribute *localID* can be used to relate the system-wide object identifiers maintained by the project data server with persistent local identifiers that may be supported (or required) by certain applications.

4.3.2 Scope and extent of the communication model data structures

The scope and visibility of the symbols contained in communication model objects is governed by the rules of the underlying Information Container schema.

Thus, each *CommunicationEvent* object is self-contained, i.e. it defines a lexical closure for all symbols used in its Information Container based representation. According to that, a reference to an Information Container object, e.g. in the *inParams* attribute of a *Request*, effectively means to embed this Information Container in the *Request* and ensure that all names involved in the embedded structure are unambiguously resolved.

The dynamic extent of the communication model objects, i.e. their existence in time, depends on their specific semantics as follows:

1) *CommunicationEvent* instances, i.e. requests and responses, exist from the time of their creation until the end of the *transaction*[*)] in which they were created;
2) A *CEESession* object is constructed either explicitly, by an *openSession* operation issued by a client (see Appendix V, Table V.4), or implicitly, by means of the constructor of the CEEsession class in the Information Container API (see Appendix II). It exists until the session is closed by an explicit *closeSession* operation or by an unexpected event, e.g. lost connection, timeout, system crash etc.;
3) *User*, *Client* and *Host* are considered quasi-static elements of the environment and are maintained persistently in the server's data store for the full duration of a project.

[*)] Transactions are not elements of the Communication Model, but special-purpose operations. They are discussed in detail in section 4.6.

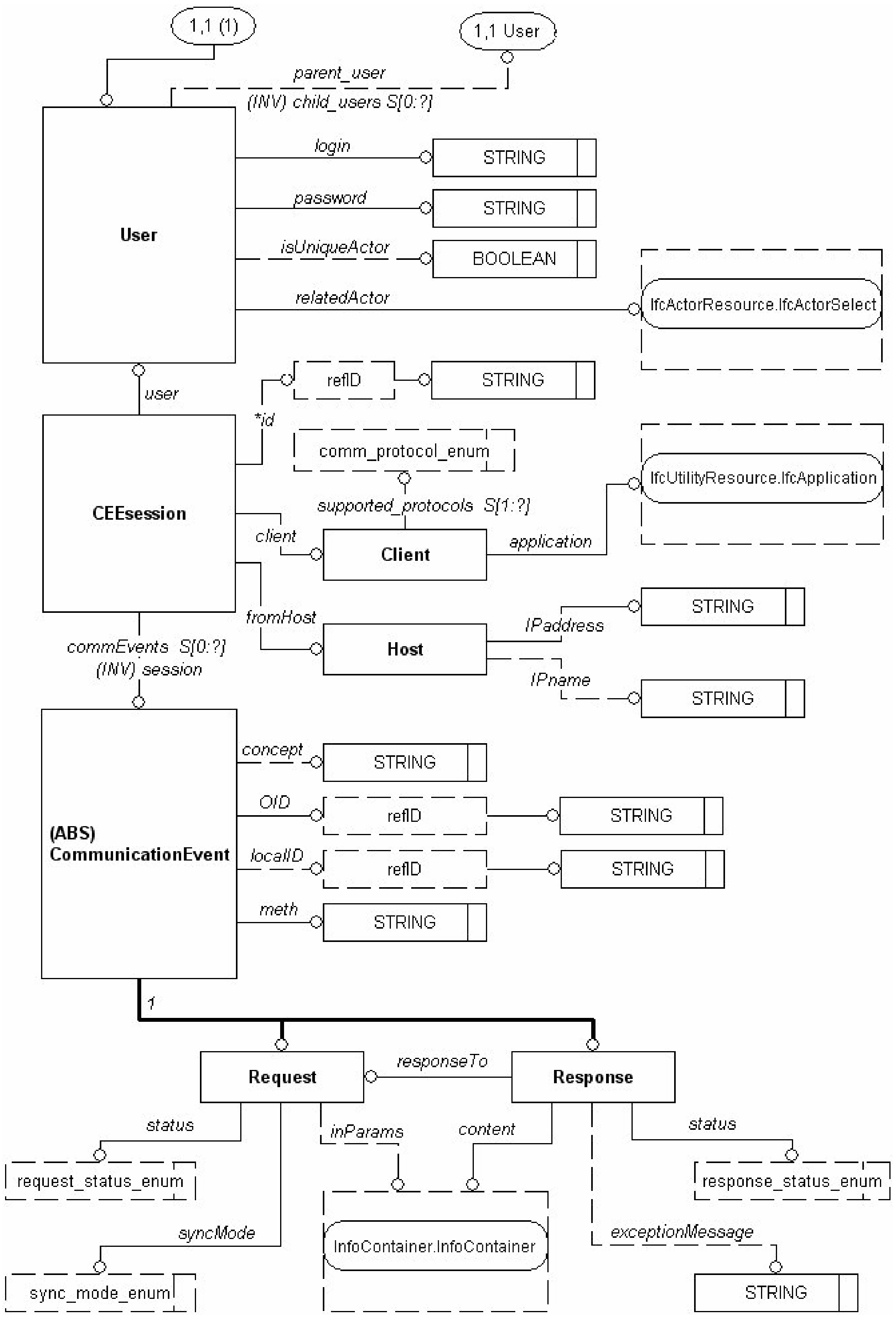


Fig. 4.4: Overview of the communication model data structures in EXPRESS-G

4.3.3 EXPRESS schema specification

The EXPRESS schema of the generic client-server communication protocol given in “short form” below, i.e. without resolution of external schema references, is intended to serve as a formal *reference model* of the involved data structures. It should not be understood as a basis for a “STEP-like” implementation, because its purpose is quite different. Thus, whilst instances of the modelling objects defined in the schema would really exist on the project data server for a particular project, they are *not* subject to data exchange or sharing with applications - neither through STEP physical files, nor through SDAI-like requests or other project data operations. Rather, they represent the *information about the information to be exchanged*, i.e. the meta data structures allowing the uniform handling of any model data in (almost) any specific project environment.

```
SCHEMA CommunicationModel;

REFERENCE FROM IfcActorResource (IfcActorSelect);
REFERENCE FROM IfcUtilityResource (IfcApplication);
REFERENCE FROM InfoContainer (InfoContainer);

TYPE comm_protocol_enum = ENUMERATION OF (CORBA, HTTP, RMI, TCP);
END_TYPE;
TYPE request_status_enum = ENUMERATION OF
  (acknowledged, deferred, executing, failed, finished, sent);
END_TYPE;
TYPE response_status_enum = ENUMERATION OF (notOK, OK);
END_TYPE;
TYPE sync_mode_enum = ENUMERATION OF (async, sync);
END_TYPE;
TYPE refID = STRING;
END_TYPE;

ENTITY CEEsession;
  id          : refID;
  user        : User;
  client      : Client;
  fromHost    : Host;
  commEvents  : SET [0:?] OF CommunicationEvent;
UNIQUE
  UR1: id;
END_ENTITY;

ENTITY User;
  login       : STRING;
  password    : STRING;
  isUniqueActor : OPTIONAL BOOLEAN;
  relatedActor  : IfcActorSelect;
  parent_user : OPTIONAL User;
INVERSE
  child_users : SET[0:?] OF User FOR parent_user;
END_ENTITY;
```

```
ENTITY Client;
  supported_protocols : SET [1:?] OF comm_protocol_enum;
  application : IfcApplication;
END_ENTITY;

ENTITY CommunicationEvent
 ABSTRACT SUPERTYPE OF (ONEOF(Request,Response));
  concept      : OPTIONAL STRING;
  OID          : refID;
  localID      : OPTIONAL refID;
  meth         : STRING;
INVERSE
  session      : CEEsession FOR commEvents;
END_ENTITY;

ENTITY Host;
  IPaddress    : STRING;
  IPname       : OPTIONAL STRING;
END_ENTITY;

ENTITY Request
 SUBTYPE OF (CommunicationEvent);
  status       : request_status_enum;
  syncMode     : sync_mode_enum;
  inParams     : OPTIONAL InfoContainer;
END_ENTITY;

ENTITY Response
 SUBTYPE OF (CommunicationEvent);
  responseTo   : Request;
  status       : response_status_enum;
  content      : InfoContainer;
  exceptionMessage : OPTIONAL STRING;
END_ENTITY;

END_SCHEMA;
```

4.3.4 Examples

The client-server request/response sequences in the two examples provided below illustrate in textual form the information exchange messages based on the definitions of the developed generalised communication model. They are taken from a simple line-oriented test client implementation using the TCP/IP communication method and the Information Container externalisation described in section 4.2. Further options of the use of different project data services in client-server communication are given in Appendix V, detailing the developed server operations.

In the shown selected examples, user input is indicated in *italics*, whereas program output, i.e. the server responses, is in regular font.

Example 1:

This example demonstrates the packaging of a simple client query and the corresponding server response into respective *Request/Response* objects. The requested operation retrieves the value of the attribute “GenericType” of a specific *IfcBeam* instance.

```
Input> Request1(session:oRef(session.1)
                OID:"MODEL.STRUCT_1:IfcBeam.232"
                meth:"getAttribute"
                inParams:InfoContainer(attName:"GenericType")
                localID:"BEAM-1"
                syncMode:sync
                status:sent)
Sync request accepted. Request name: "Request.43" .

Server response:
Response1(
  responseTo:oRef(Request.43)
  status:ok
  content:outParams(attName:"GenericType" attContent:TRUSS)
  session:oRef(session.1)
  OID:oRef(MODEL.STRUCT_1:IfcBeam.232)
  localID:"BEAM-1"
  meth:"getAttribute")
```

Comment:

As the example shows, the server response is returned in the form of an Information Container in accordance with the definition of the *Response* entity in the EXPRESS specification of the communication model. If needed, the client application can subsequently map this Information Container to some other format*).

The response is somewhat verbose, but this can be useful for faster parsing of the data by the client.

Example 2:

This example demonstrates the retrieval of a full product model from the server as a STEP physical file. This is done in asynchronous mode because the operation is assumed to take longer time to execute.

To illustrate the procedure in more explicit form, a sequence of five inter-related requests is presented. The first request is to create a STEP physical file for the specified model on the server, the second and the third query the execution status of the first, the fourth fetches the stored result of the “retrieve” operation from the server, and the last performs the actual file download.

*) For the case that the client application is written in Java, this can easily be done with the help of the object classes of the Information Container API for Java presented in appendix II.

```
Input> Request1(session:oRef(session.2)        -- issues the async  request
                OID:"MODEL.STRUCT_1" meth:"retrieve"
                syncMode:async
                status:sent)
Async request accepted. Request name: "Request.44" .
Input> Request2(session:oRef(session.2)        -- queries the request status
                OID:"Request.44" meth:"getRequestStatus")
Sync request accepted. Request name: "Request.45" .
Server response:
Response2(
  responseTo:oRef(Request.45)
  status:ok
  content:outParams(requestStatus:executing)
  session:oRef(session.2)
  OID:oRef(Request.44)
  meth:"getRequestStatus")
Input> Request3(session:oRef(session.2)   -- repeat query until status=finished
                OID:"Request.44" meth:"getRequestStatus")
Sync request accepted. Request name: "Request.46" .
Server response:
Response3(
  responseTo:oRef(Request.46)
  status:ok
  content:outParams(requestStatus:finished)
  session:oRef(session.2)
  OID:oRef(Request.44)
  meth:"getRequestStatus")
Input> Request4(session:oRef(session.2)               -- get the result
                OID:"Request.44" meth:"getResponse")
Sync request accepted. Request name: "Request.47" .
Server response:
Response4(
  responseTo:oRef(Request.47)
  status:ok
  content:outParams(repositoryRef:BLOB(STRUCT_1))
  session:oRef(session.2)
  OID:oRef(Request.44)
  meth:"getResponse")
Input> Request5(session:oRef(session.2)    -- download the referenced BLOB
                OID:"MODEL.STRUCT_1"
                meth:"download"
                inParams:ic(repositoryRef:BLOB(STRUCT_1)))
Sync request accepted. Request name: "Request.48" .
Server response:
Response5(
  responseTo:oRef(Request.48)
  status:ok
  session:oRef(session.2)
  OID:oRef(MODEL.STRUCT_1)          user input: local file on the client system
  meth:"download")                                   ⇩
Local file name for storing BLOB(STRUCT_1): c:\test\s1.spf
Downloading BLOB(STRUCT_1) ... done.
```

Comment:

The presented request/response sequence is more complicated compared to the first example because of the first asynchronous request. In the following synchronous requests/responses the communication mode is not specified explicitly as "sync" is the default mode. The request status is also not specified as it defaults always to "sent". This feature is provided primarily to enable querying the execution status of *asynchronous requests*, with possible server responses "acknowledged", "deferred", "executing", "finished" and "failed" (see requests 2 and 3).

The last shown request differs from the normal procedure. This is seen at the server response as well, which does not contain any detailed output. In fact, since the server is not allowed to write directly to a local file on the remote client, it sends the requested file to an additional socket stream reserved for BLOB upload/download. It is on the responsibility of the client application to read this stream and store the data in a file. The request itself is only necessary to obtain the required permission to download the prepared data*).

4.4 Object-Oriented Project Data Operations

Operations are the third fundamental information representation concept provided at the ontology level of the environment. Without explicitly defined public operations, the exchange and sharing of data will still be possible, but in a form in which the model data will be independent of the applications, and the applications will be “external” to the data models, “*inhibiting the 'smart' participation of the data models in the process of project realisation*” (Wittenoom 1998).

In order to enable the implementation and the use of intelligent capabilities at the project data server managing the data models of the CEE framework for all design disciplines and all involved applications, in this section an approach for the specification and implementation of operations is proposed that allows to:

1) Provide a high-level abstraction of object-oriented server methods, which is compatible with distributed object technology, as proposed by CORBA (Orfali et al. 1997) or Java RMI (Harold 1997), but is also applicable to non object-oriented design tools;
2) Align the representation of operations with the representation of Information Containers and the developed generalised communication model, ensuring the overall coherence of the components of the ontology;
3) Align the definition of operations with the specification of EXPRESS data models to enable common understanding on the level of model development and specification.

In fact, the use of operations was already silently introduced in the two examples given in the previous section, but the principles of their specification, implementation and usage by client applications need yet to be explained.

*) This procedure is much easier to implement by Java RMI based communication with the help of the BLOB method *saveAsFile*, and is automatically provided by WWW-Browsers.
For file upload, done in a similar way, the BLOB methods involved are *getInStream* and *loadFromFile* (see Appendix II).

The key ideas of the suggested approach are as follows:

1) Operations are not defined by a separate representation formalism, but utilise the specifications provided in the Meta model, the Information Container and the communication model schemas. On the high-level, i.e. in the Meta model, they are defined generically, as part of the meta class *Concept*, and thus are available for any object in any data model.

2) The specific operations associated with the representations of the particular classes of a data model supported by the project data server are specified with the help of a subset of the EXPRESS-C notation developed in the European PISA project (cf. ISO/TC184/SC4/WG5/N202 1994)[*)]. This enables the language-independent specification of object methods, ensures conformance with the EXPRESS data models, allows formal parsing of the models including the defined operations[**)], and provides a basis for language bindings for the implementation of distributed object processing in the sense of CORBA.

The definition of the arguments of an operation is done in one of the following two ways: (1) by using basic EXPRESS data types and the components of the respective EXPRESS data model, or (2) by using the components of the Information Container schema.

The first method is more specific and assumes that all applications that will use the operation will have the same “understanding” of the referenced data structures. The second method is more generic and assumes that the input and the output of the operation will be “interpreted” by the application in accordance with its specific functionality and the specific context in which the operation is executed, i.e. the particular object instance being addressed[#)].

3) The realisation of the EXPRESS-C specifications in the CEE system is achieved according to the principle of *method delegation* (see fig. 4.5).

Method delegation establishes a regime of object-to-object communication allowing one object to have enhanced capabilities by acting as an extension of another object (cf. Zucker & Demaid 1992). With the approach suggested herein, it is possible to extend these capabilities across platforms and applications. For example, a structural analysis application containing data structures for the representation of linear building elements, but not for beams and columns, may delegate the task of providing the cross section properties of its linear building elements to the “beam” and “column” objects on the project data server. Another possibility, which has been realised in the prototyped environment, is to implement a high-level operation for the geometric presentation of all tangible objects in an application, which is then delegated to the server data objects that may even not coincide with the objects of the client application.

[*)] The EXPRESS-C signature of an operation consists of the operation name, along with the specification of input and output arguments. Output arguments are distinguished by the keyword VAR, optional arguments are denoted by the keyword OPTIONAL.

[**)] For example, with the help of the ECCO toolkit developed at the University of Karlsruhe in the frames of the PISA project.

[#)] This “more generic” method has been used for all operations in the prototype implementation of the project data server presented in chapter 8.

This functionality is accomplished in the following way:

First, a generalisation of the attributes of an operation to Information Container data structures is made on the ontology level. In this way both the uniformity of the representation across applications and data models, and the use of operations by these applications, regardless of their internal data structures, is achieved.

Secondly, the object methods corresponding to the defined EXPRESS-C operations are implemented on the project data server, using its native program structures. The input and output parameters are provided according to the generalised operation signatures in terms of appropriate Information Container data types. For this purpose, generic "wrapper" code, simplifying the software realisation, has been applied.

The implementation of each operation may take full advantage of the object-oriented structure of the data models and the generalised form of the operation specifications. Thus, it is possible to define polymorphic methods on high-level, specialising them in specific lower level objects. A typical example of such implementation is provided by the operation "view", which is defined with an empty body at the level of the abstract *IfcProduct* object and is then specialised to retrieve the geometry of any "product" object in appropriate way. This method is utilised by the "view" operation working at model level, which simply scans the model data and applies consecutively the *IfcProduct* "view" operation to all "product" objects found.

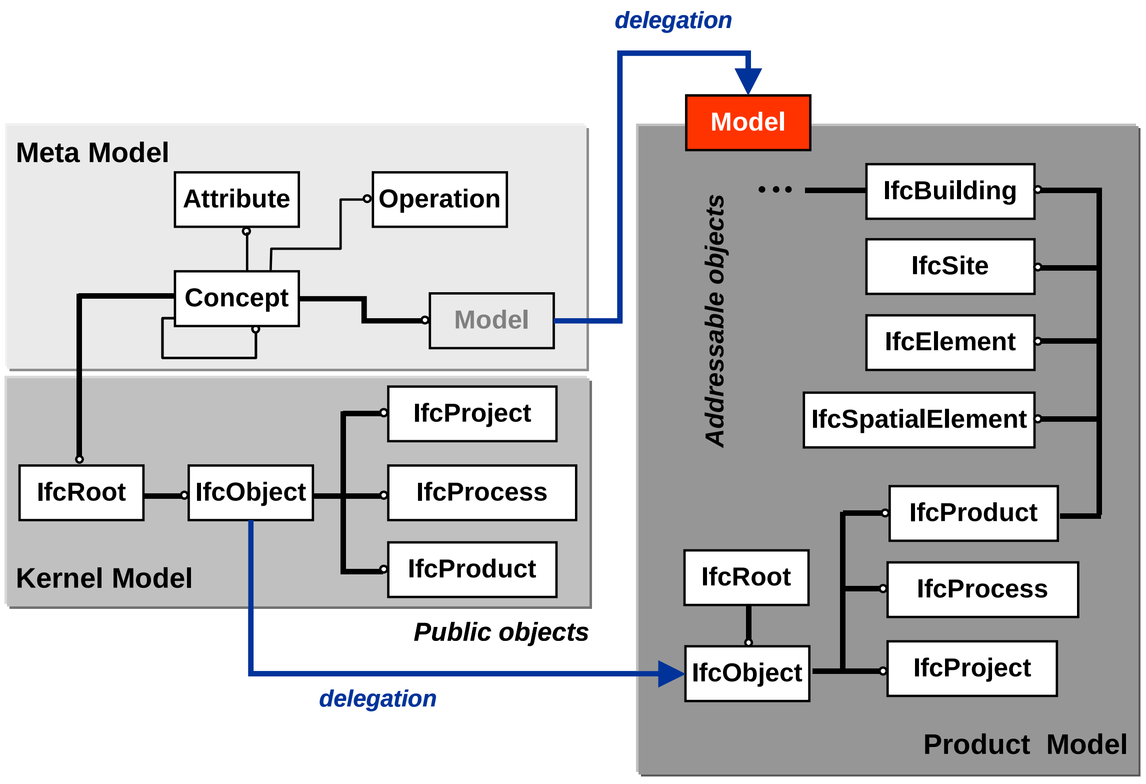


Fig. 4.5 Achieving the specific functionality of public objects by method delegation

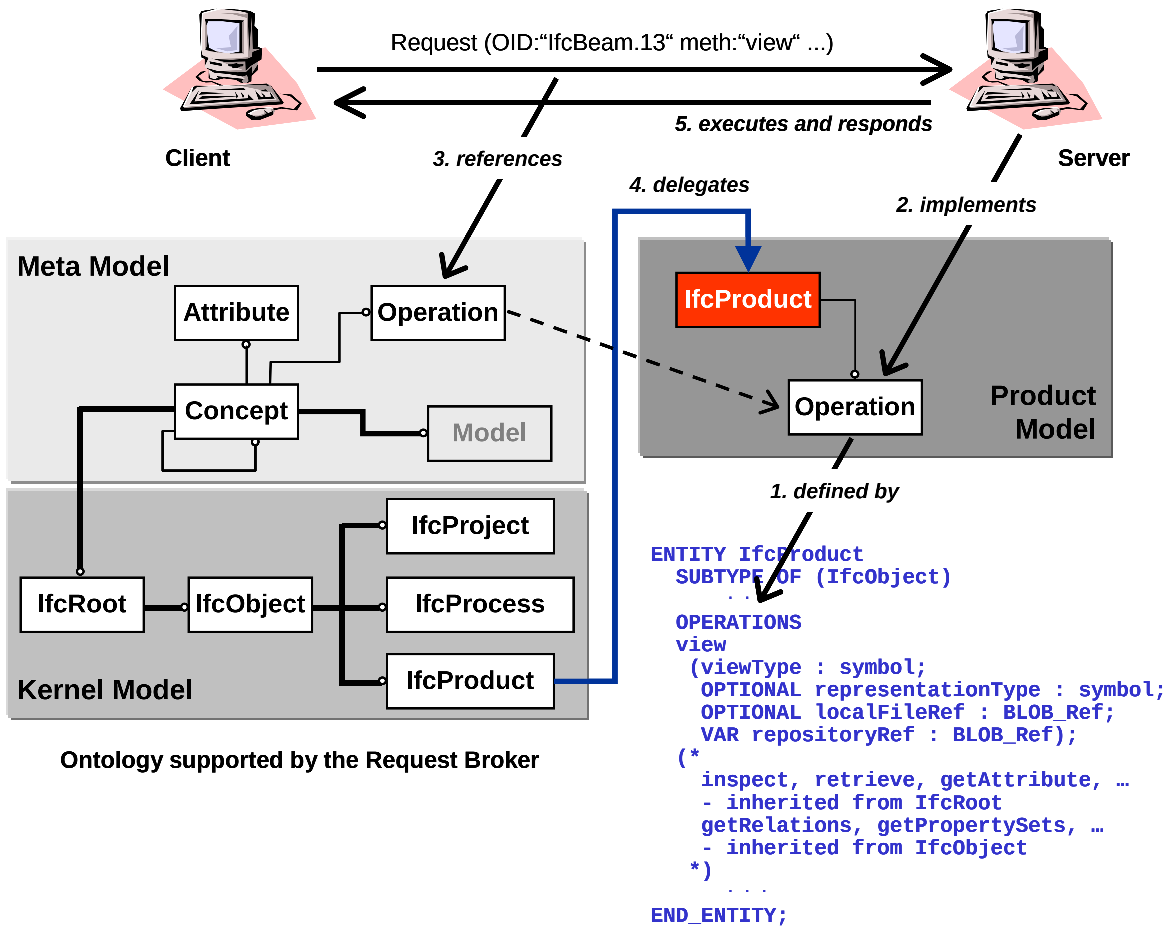
4) At run-time, a server method can be invoked by client applications either *indirectly*, by using the procedure described in the previous section, i.e. by embedding it in a respective *Request*, with the object reference provided in the *OID* parameter of the request, the method name provided in the *meth* parameter, and the method parameters packaged in an Information Container; or *directly*, by using remote method invocation. In the latter case, there are two further alternatives:

– if the client application implements internally a different object structure, a realisation with the help of Information Containers will be necessary, along with pre- and post-processing of the sent/received data;
– if the client application implements internally the same data object(s) as the server, and does not require any conversion utilities, a “true” object-oriented realisation in the sense of CORBA will be possible.

A rough analogy of these alternatives to the data access techniques developed for relational databases is the use of SQL, embedded SQL and ODBC respectively (cf. Lockemann et al. 1993).

However, in all cases the top-level parameters of a request need to be provided, as they ensure the authorisation to execute a particular operation (user, role), the mode of the operation (synchronous, asynchronous), and the correct performance of update operations in client/server transactions.

Fig. 4.6 schematically illustrates the overall approach, and provides an idea of the realisation procedure.



```
ENTITY IfcProduct
  SUBTYPE OF (IfcObject)
    . .
  OPERATIONS
  view
   (viewType : symbol;
    OPTIONAL representationType : symbol;
    OPTIONAL localFileRef : BLOB_Ref;
    VAR repositoryRef : BLOB_Ref);
  (*
    inspect, retrieve, getAttribute, …
    - inherited from IfcRoot
    getRelations, getPropertySets, …
    - inherited from IfcObject
   *)
    . . .
END_ENTITY;
```

Fig. 4.6: Schematic presentation of the specification, implementation and usage of operations

Operations may be developed on different levels of the modelling framework (kernel, neutral, domain models). However, most important to the overall data management capabilities of the CEE system is the definition of operations on the Meta Model level, and especially for the meta object ***MODEL***, providing the requisite meta information needed to manipulate models as a whole.

This is due to the following reasons:

– the assumed structuring of the information in a set of inter-linked models, enabling the implementation of a private workspace for each designer, and not in a single large shared model;

– the prevailing operations on large portions of the data, such as a whole building, a building storey, or a building system, which are more conveniently specified on model level;

– the typical use of CAD systems where the transfer of whole models by means of data exchange files is most commonly practised, and

– last but not least, the specific character of the design process, requiring long periods of work on one and the same data set, as opposed to the short transactions typical by business databases.

Fig. 4.5 provides a hint that operations can be delegated to a model as a whole. In fact, most such operation are valid for any model, because the specific underlying schemas are not relevant for the specification of generic model-level operations.

Of course, operations can be defined also for any object class in any of the data models "known" to the project data server. However, depending on the *type of the object class w.r.t. its operations*, their use may be different.

In principle, irrespective of the object classifications provided in the data model schemas, the following three categories of objects can be identified: (1) registered public objects, (2) addressable objects, and (3) private objects.

Registered public objects are all objects that are known to the Request Broker and are part of the system ontology. They can be accessed directly by their object IDs, and all their operations are visible to all client applications (provided that there is no restriction in the access rights). Operations on such objects can be performed explicitly, by issuing respective requests, or by using remote method invocation.

In contrast, *addressable objects* are objects that are not known to the Request Broker and are not part of the system ontology. They can be accessed by applying method delegation. This is accomplished by first obtaining their *reference IDs* from the project data server, and then using these IDs in subsequent operations*). Remote method invocation is only possible through the *execMeth* method of the object reference. Another principal difference between addressable and public objects is that public objects are persistently maintained during a client/server session and may be accessed repeatedly, without affecting the state of the model data in the data store, whereas addressable objects are retrieved anew by each single operation.

However, there is no conceptual restriction in the functionality of addressable objects compared to public objects. The rationale for the introduction of this category is to reduce

*) A detailed example of this technique is presented in appendix II, section II.3.

the computational load of the Request Broker w.r.t. name resolution, to provide a structured method for the definition of access rights, and to separate application domains. This is especially important for an environment comprised of hundreds of object classes, structured in a number of layered data models.

At last, *private objects* are all objects that cannot be accessed directly through operations issued by client applications. Such objects may still possess methods that serve other objects, but these methods can only be used implicitly, as side effects of other operations.

The classification of the object classes in the data models in accordance with the above aspects is an implementation feature. It is done on meta level and does not affect the public EXPRESS-C specifications provided in the model schemas. However, this classification can be undertaken in different ways, and it does affect the functionality of the system.

As an example, the classification of IFC object classes which has been assumed for the prototyped CEE implementation is presented in table 4.1. This classification is very generic, with only a few object classes declared as public. It leads to a lean Object Request Broker, shifting much computational load to the clients. Obviously, for a practical CEE system, using specific project data models, such as IFC, additional considerations, tests and experimentation may be needed to achieve optimal performance.

Table 4.1: IFC object types with respect to systemic interoperability

Registered public objects	Addressable objects	Accessible objects
Meta model: Concept, Model, ModelSchema, MappingSchema **Kernel model:** IfcRoot, IfcRelationship, IfcRelGroups, IfcObject, IfcProject, IfcProduct	All other objects of the Kernel Model, and all objects in domain-specific models (architectural, structural, HVAC etc.)	All resource objects and all application schema objects

4.5 Project Data Services

On the basis of the general approach for the specification of operations, any kind of functionality can be provided. In principle, even operations like

```
doStructuralAnalysis (forModel: Obj_ref, loads: BLOB_Ref,
        controlData: BLOB_Ref, output: BLOB_Ref, mode: String);
```

can be imagined.

However, this should be done in a structured way as well, in order to enable modular implementation of the project data server and to separate basic server operations that are more closely associated to data management aspects from advanced services that can be more appropriately implemented as *stored procedures* (cf. Lockemann et al. 1993), or even as *knowledge-based agents* (cf. Russel & Norvig 1995), complementing the basic server functionality.

For this purpose, the following categorisation of project data services is proposed:

(1) general data management services on object level;
(2) general model management services;
(3) specific services for access control;
(4) interoperability services;
(5) services for session and transaction management;
(6) advanced services.

The first five categories are realised as *basic operations* in accordance with the principles presented in the previous section. However, albeit being categorised as basic, some of these operations are not at all simple. For example, the *map* operation included in the fourth category requires complex server actions and heavy computational resources, and may take a lot of time to execute. Nevertheless, it is a basic operation as it is directly related to the management of the data in the environment, and does not require specific engineering knowledge for its practical realisation.

The key idea of the first category is to provide read/write access to individual object instances and their attributes on the basis of short transactions.

The second category is proposed for the management of the data in a model as a whole, including such features as checking in/out the model, retrieving different presentation views etc.

The third category provides the necessary operations to maintain access control lists (ACL) for modelling objects and operations, i.e. to create and modify “users” and “roles”, and to set, unset and query their respective access rights.

The fourth category is dedicated to sophisticated operations for the maintenance of the integrity and the consistency of the project data, as well as for facilitating user-driven coordination and reconciliation processes.

The fifth category includes control operations. Their purpose is not to examine or change the data in the repository, but to provide facilities for the management of the communication process itself.

The last category includes, conceptually, stored procedures and agents that can be of arbitrary complexity. Whilst the realisation of such services has not been the intention of this work, I believe that with the provision of a set of basic operations, they can be easier achieved as they can use the basic operations as “building blocks”, and implement all more sophisticated functionality on top of them, without bothering about low level data management.

Before the execution of all suggested basic operations the following procedure is applied:

– first, the user access rights are checked;
– second, depending on the access rights and the model state, the availability of the data is determined and compared to the type of the requested operation.

The operation is performed only if these two checks are passed successfully.

Table 4.2 below presents a list of basic services suggested for CEE. Detailed specifications of the operations implemented in the prototyped environment are provided in appendix III. Model states and access rights are discussed in the following section.

Table 4.2: Basic project data services

Category	Suggested Operations
General data management services on object level	Create instance, Retrieve, Update, Inspect, Find, Get/set/unset/test attributes, Get instances, Get relations, Get methods, Get Status, View, Group, Ungroup, …
General model management services	Create model, Delete model, Retrieve model, Check in, Check out, Find in model, Get model objects, Get product objects, Get views, Get groups, Get versions, Status, View, …
Specific services for access control	Create/remove user, Create/remove/assign role, Get/set/unset access rights, …
Interoperability services	Map models, Match model versions, Check consistency, Merge models, GetMappingSchemas, Get related documents, …
Services for session and transaction management	Open/close session, Begin transaction, Commit, Rollback, Abort, Get request status, Get Response, …

4.6 Transactions and Model States

Most of the above object-oriented operations will access the data repositories of the project data server, possibly attempting to update its content. Thus, from the point of view of information sharing these operations can be seen as elementary **transactions**.

In database terminology a transaction is usually understood as a logical block of one or more elementary operations, which appears as one single operation w.r.t. the data store. Normally, the packaging of operations is related to performance aspects as it can help reduce the client/server interactions and optimise the actual data access routines.

In the context of the proposed systemic interoperability approach the concept of transactions is even more important as it enables the realisation of more sophisticated services on top of a potentially large number of basic operations.

All transactions are characterised by the following four features, also known as their ACID properties: (1) atomicity, (2) consistency, (3) isolation, and (4) durability (cf. Gray & Reuter 1993).

Atomicity means that all operations packed in a transaction should be seen as one single operation. The transaction is successful if and only if they are all successfully completed.

Consistency means that a completed transaction should not violate the consistency of the data store.

Isolation means that if a transaction needs to be undone (rolled back), there should be no other transactions that would have to be undone as well, i.e. the transaction must not have any inter-dependencies with other transactions[*)].
At last, *durability* means that the changes of the data store caused by a transaction must remain persistent after the transaction, and should not be influenced by system crashes and/or recovery operations from system crashes.

These basic features of client/server transactions have to be observed by the execution of all operations. However, as already mentioned in section 4.1, in the envisaged CEE system it would be necessary to support **long transactions** related to the transfer of large amounts of data and their processing over long periods of time.

Unlike short transactions executed within a single client/server session, long transactions are characterised by an asymmetric two-step check-in/check-out procedure and require the maintenance of appropriate *long duration locks* (cf. Herrmann 1991; Dadam 1996).

There is no principal difference between a “normal” read lock (R) and a long read lock (LR), but the semantics of write locks is not the same. Whilst a “normal” write lock (W) locks the data during the whole update operation, in a long transaction the update is split into two parts. When a client requests a change of the data, it acquires a long write lock (LW), and the data is copied to a private workspace accessible only to that client. The LW lock is automatically converted to an exclusive check-in lock (CI) when the data is checked in again. However, even with this improvement of server responses to requested long duration transactions, the procedure is still very restrictive for the purposes of cooperative, concurrent work. As shown in table 4.3, it allows quite a few types of simultaneous operations, and does not provide a method for concurrent changes of the data. Thus, if a designer locks a whole building by a long write transaction, no other designer will be able to modify any of the building data before the first designer checks it in again. Concurrent access will be inhibited for a number of days, and there will be no possibilities for creating and comparing alternative solutions.

Table 4.3: Compatibility matrix of model locks / after (Dadam 1996) /

Lock type	R	W	LR	LW	CI
R	+	-	+	+	-
W	-	-	-	-	-
LR	+	-	+	-	-
LW	+	-	-	-	-
CI	-	-	-	-	-

Key: R = Read Lock; W = Write Lock; LR = Long Read Lock;
LW = Long Write Lock; CI = Check In Lock

*) This property is especially important for the serializability of parallel, overlapping operations that can be very frequent in the case of long transactions.

In order to tackle such issues, the concept of **model states** and **versions** is introduced. The proposed method for dealing with the problems of long transactions is principally valid for individual objects as well, but its application is intentionally suggested only on model level. This is due to the following reasons:

– theoretical and practical investigations of most of the aspects related to long transactions in engineering databases are still in their infancy (cf. Dadam 1996); it is therefore useful to first examine simpler cases;
– the management of versions on object level is much more complicated than on model level, which makes it a research issue in its own right (cf. Sieberer & Keber 1997);
– typical design tasks update the model data in large batch transactions, affecting many of the objects they contain.

The essence of the proposed method is as follows:

The defined operations (which are in fact also objects) are classified in the following 12 categories: create model, delete model, open model, close model, (short) read, (short) write, long read, long write, rollback, commit prepare, commit and merge. In addition, there are also control operations that set temporary exclusive locks, but do not affect the model's state.

Model states and the respective model locks are extended by three new types: *Ready to commit* (RC), *Commit* (COM) and *Merge* (MRG)[*)].

This leads to the state transition diagram presented on fig. 4.7 below.

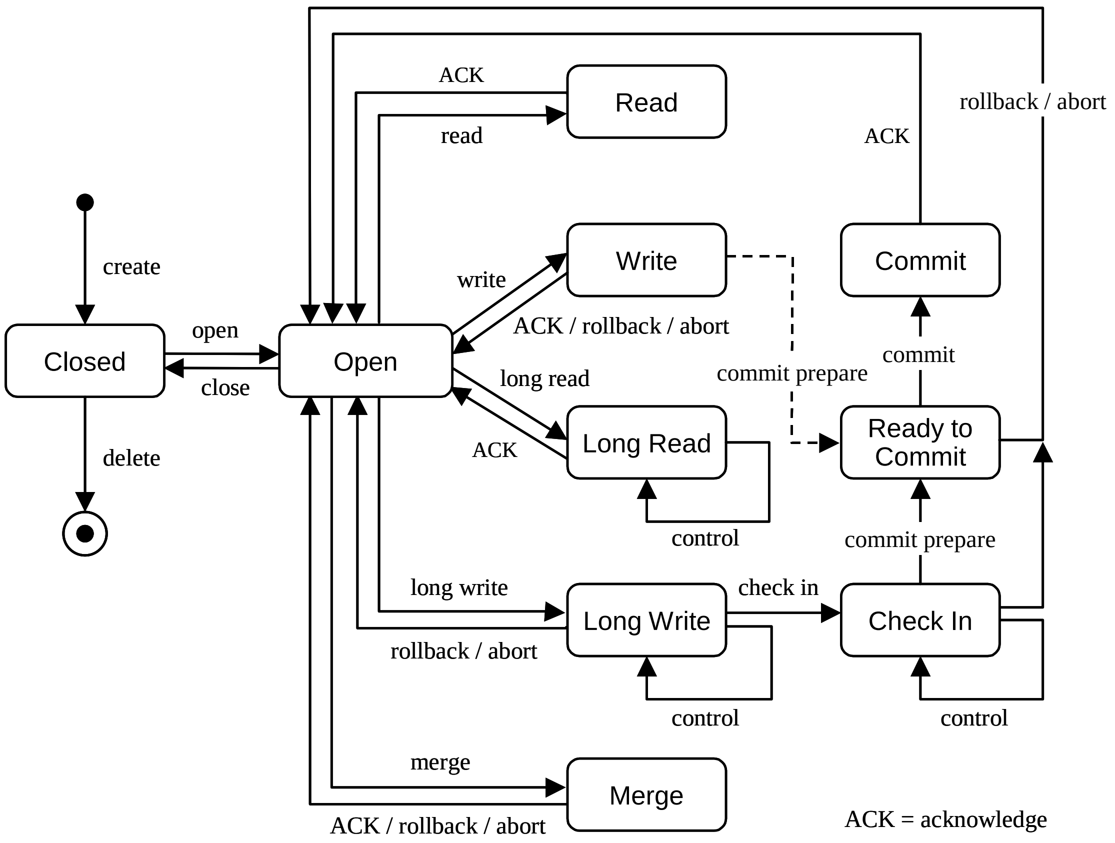


Fig. 4.7: Model states and principal operations causing state transitions

*) *Closed* and *Open* are self-explanatory and are presented only for completeness.

Whilst by themselves these proposed new model states and locks do not provide additional flexibility for concurrent model updates, they become important when model versions and user access rights are considered as well.

The **user access rights** are maintained in respective access control lists (ACL), as meta data associated to the models. Each model is assigned a *role*, for example “architect”, and each role is associated with one or more users. Of these users, there is exactly one declared as *owner* of the model, whereas all others are “normal” users with standard access rights. In addition, *superuser* access rights are assigned to the system administrator or information manager, providing him with the privilege to access and update all data at all times.

The access rights themselves are similar to the typical access rights maintained by operating systems, i.e. “read”, “write” and “execute” (enabling/disabling the execution of operations).

The rationale of defining specific owner rights is to provide exactly one user with the ability to execute short write transactions (W), which are reflected directly in the model data, and do not lead to new model versions. In contrast, all other users may at most have the right to execute long write transactions, which are always associated with the generation of new model versions.

Update operations are not executed immediately, but are explicitly committed by using a “safe” *two phase commit protocol* (2PC) adopted from distributed database technology (Özsu & Valduriez 1991; Gray & Reuter 1993). However, whilst in distributed databases 2PC-protocols are usually used to ensure the success of global transactions where one client simultaneously updates data in more than one databases on more than one servers, here the idea is to avoid inconsistencies because of possible system crashes during “long updates” associated with the generation of new model versions, with the merging of two or more model versions, or with the linking of all domain-specific models for the creation of a new, consistent project data context.

According to that approach, a model that is checked out for modification, i.e. changing the properties of the design objects, is first put in long write state (LW). When checked in again, the LW lock is released, and a new model version is created. Thus, the result of a *check in* operation are two model versions: the old (unchanged) model version, marked as “open”, and the new (changed) version, marked as “checked in”. As indicated on fig. 4.7, both these versions are accessible to a number of concurrent operations. However, after a *commit prepare*, the checked in model is provided with an exclusive RC-lock, disabling any other operations except *commit* and *rollback*. On the server, the model data is linked with all models related to it in a temporary workspace. The *ready to commit* operation is acknowledged if and only if the update is successful. Thus, whilst the client still “believes” that the data is not committed, the update operation is executed already before the *commit* request is made. In this way, the actual commitment is simplified to local adjustment of the version names and the model locks by the server, which minimises the risk of failure.

In addition, different versions of one model can be matched to discover the differences in the modelling objects, or they can be manually reconciled and then merged to obtain a new overall consistent project model version.

In this way, much greater possibilities for concurrent access are provided, as shown on tables 4.4, 4.5 and 4.6.

The properties of the prototyped operations with respect to model states and permissible execution modes are detailed in appendix V, table V.5.

Table 4.4: Compatibility matrix of permissible concurrent operations by the application of model versions

Lock type	R	W	LR	LW	CI	RC	COM	MRG
R	+	-	+	+	+	+	+	-
W	-	-	-	-	-	-	-	-
LR	+	-	+	+	+	+	+	-
LW	+	-	+	+	+	+	+	-
CI	+	-	+	+	+	+	+	-
RC	+	-	+	+	+	-	-	-
COM	+	-	+	+	+	-	-	-
MRG	-	-	-	-	-	-	-	-

Key: R = Read Lock; W = Write Lock; LR = Long Read Lock; LW = Long Write Lock; CI = Check In Lock; RC = Ready to Commit Lock; COM = Commit Lock; MRG = Merge Lock.

Table 4.5: Compatibility matrix of permissible concurrent operations on different models

Lock type	Model B ⇨	R	W	LR / LW / CI	RC	COM	MRG
Model A ⇩							
R		+	+	+	-	+	-
W		+	+	+	-	-	-
LR / LW / CI		+	+	+	-	+	-
RC		-	-	-	-	-	-
COM		+	-	+	-	-	-
MRG		-	-	-	-	-	+

Note: All abbreviations are the same as in table 4.4.

Table 4.6: Permissible initial model states for the different operation types

Operation type	Permissible model states for the execution of the operation
Read	Open / Read / LongRead / LongWrite / CheckIn / Commit
Write	Open
LongRead	Open / Read / LongRead / LongWrite / CheckIn / Commit
LongWrite	Open / Read / LongRead / LongWrite / CheckIn / Commit
CheckIn	LongWrite
ReadyToCommit	Write / CheckIn
Commit	ReadyToCommit
Merge	Open
Control op.	Open / Read / LongRead / LongWrite / CheckIn

In the prototype implementation of the project data server, transactions are realised through the operations *beginTransaction*, *commitPrepare*, *commit* and *rollback*. In addition, in order to simplify client implementation, a *background transaction* is introduced, i.e. all operations which are not enclosed in a *"beginTransaction-commit/rollback"* block are treated as background operations which are implicitly committed at once.

A comprehensive treatment of model versions has not been developed, but methods have been provided to facilitate *version reconciliation*. These methods include formal light-weight consistency checking of the model data, model matching and model merging. They are briefly discussed in chapter 8.

4.7 Extension of the Generalised Client-Server Communication Model with Knowledge-Based Queries and Assertions

Many specific concurrent engineering tasks can be tackled more efficiently with the application of advanced knowledge-based methods built upon the basic object-oriented project data services, as for example model mapping and matching (Khedro et al. 1994; Katranuschkov & Scherer 1997), building code services (Han et al. 1999), information searches in the full project space (Scherer 1998b) etc. These tasks are difficult to achieve by using pure object-oriented technology because of the following restrictions of the object-oriented modelling approach (cf. Woods 1991, Hakim & Garrett 1994, Borgida 1991, 1995):

- the lack of automatic classification mechanisms,
- the limited ability to represent partial object descriptions,
- the static representation of object-to-object relationships which does not allow to deduce indirect "links by value",
- the limited representation and checking capabilities for integrity and consistency constraints, and
- last but not least, the lack of deductive reasoning capabilities.

Needs for these missing features typically occur in the early design phases where object evolution has to be considered, and the modelling objects are often not fully defined.

Therefore, the implementation of a project data server for concurrent engineering support needs to incorporate more advanced representation methods, such as rule-based reasoning or description logic.

However, in order to accommodate knowledge-based processing in an inherently object-oriented modelling environment, it is necessary to provide an adequate representation formalism that can be embedded both in the object-oriented server methods and in the client-server information exchange based on the described generalised communication model. In this context, the requirements for this formalism can be defined as follows:

1) It has to provide possibilities to use the knowledge-based features of the server by means of queries and assertions that can be easily constructed and interpreted even by non knowledge-based applications;
2) It has to enable querying and filtering of model data according to ad hoc criteria which may not be explicitly available in the underlying data structures;
3) It has to be consistent with the Information Container model to ensure the integrated use of object-oriented and knowledge-based methods in a uniform manner;
4) It has to be simple so that existing applications can be easily upgraded without much development effort.

This section presents the specification of the proposed knowledge-based extensions to the generalised communication model developed in correspondence with the above requirements as part of this thesis.

Its basic idea is in the definition of a set of parameterised knowledge-base templates that can be used as building blocks for the assembly of more complex queries and assertions. The main emphasis is on the definition of search expressions that can be embedded in object-oriented operations to return “normal” object-oriented data structures that can be readily processed by object-oriented applications.

4.7.1 Structure and components of the proposed knowledge-based extensions

Fig. 4.8 below presents the top-level structure of knowledge-based expressions intended to be embedded in client requests. Its basic element is the concept of ***search expression***, which is associated with a *search template* and a set of one or more variables that accommodate the result of the search.

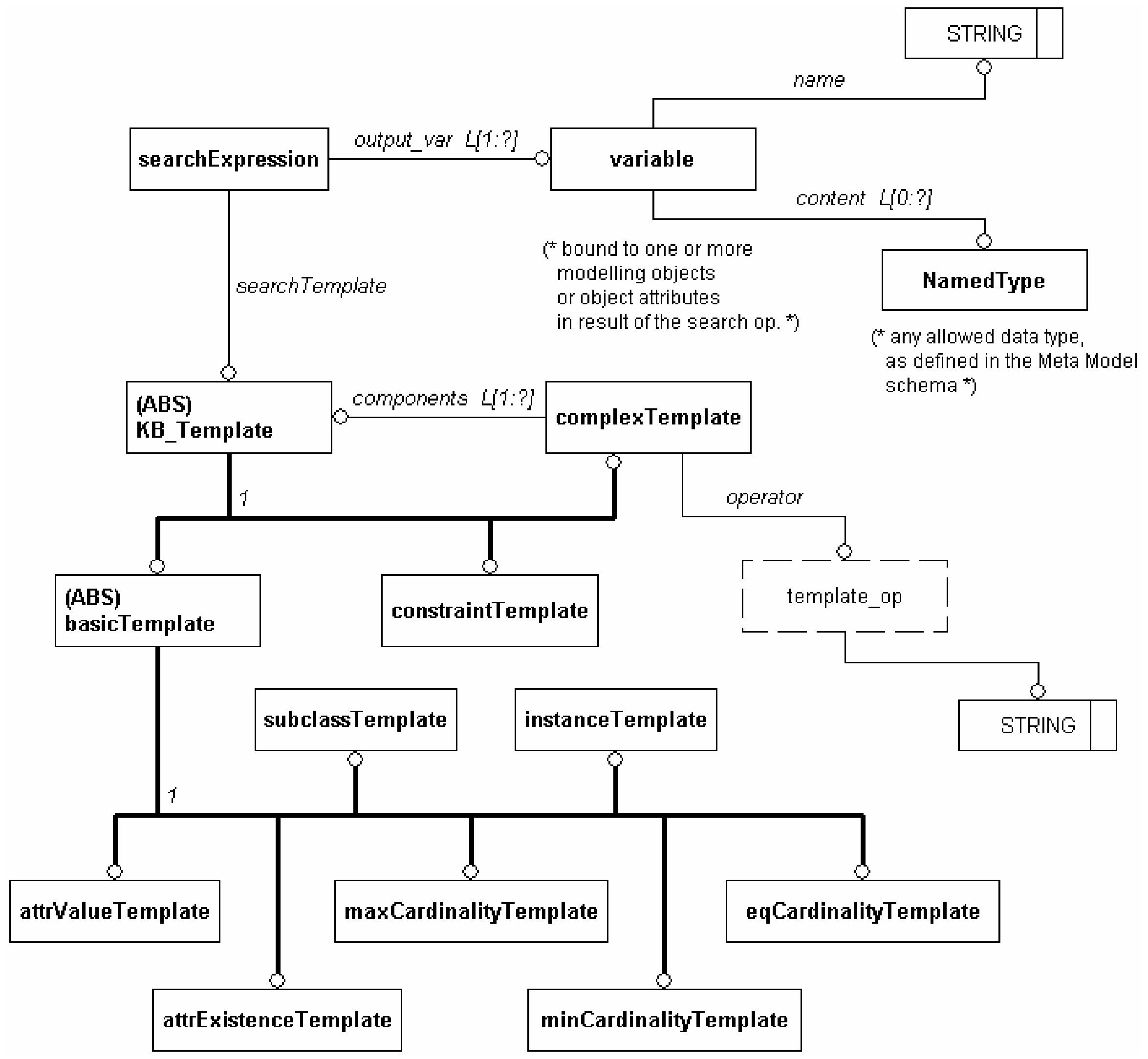


Fig. 4.8: Overview of the high-level data structures for the proposed knowledge-based extensions in EXPRESS-G

Knowledge-base templates are directly related to the structure of the underlying EXPRESS data models. However, they present additional means to query these structures with the help of an inference engine implemented by the project data server. In this way, relationships that are not explicitly represented in a data model can be derived.

To achieve that functionality, a set of ***basic templates*** is defined. These templates act as logical predicates enabling to determine if a data object, dynamically bound to a specified free variable, is a subclass or an instance of a given class, if it defines a specific attribute, if the value of this attribute is set, or if it is equal to a given constant value. In addition, three cardinality templates provide a capability to check if an aggregate attribute (list, set) contains more, less or equal number of elements compared to the value of a given integer constant.

These basic templates can be used to construct ***complex templates***, comprised of one or more component (basic or complex) templates and linked by a template operator.

The purpose of the ***template operators*** is to combine the results obtained by the individual templates. The ALL operator indicates that each template has to evaluate to true for all tested objects, and NONE indicates that all related templates have to evaluate to false for all tested objects. NOT, AND and OR represent the usual logical expression connectors.

At last, a special ***constraint template*** enables the comparison of the values of two or more templates by means of relational operators (**=** , **<** , **>** , **<=** , **>=** , **<>** , **:=:**). The last of these operators, **:=:**, checks the "identity of two data structures, whereas the first checks their equality, in a similar way as the Common LISP functions `eq` and `equal`, respectively.

The syntax of the knowledge-base templates is partially borrowed from the TellAndAsk language of the KEE system (Intellicorp 1994). The actual prototype implementation of knowledge-based queries and assertions uses the inference engine provided by KEE.

4.7.2 Syntax Specification

The syntax specification for knowledge-based expressions is comprised of three parts: (1) *tokens*, providing low-level definitions for the allowed lexical elements, (2) *grammar rules* presenting all high-level structures, and (3) *informal propositions*, including additional information not covered in the formal specifications for conciseness, or because it presents imported constructs described in other sources. The primary production rule of the syntax specification is rule (64).

Tokens

The following productions define the tokens used for the specification of knowledge-based expressions in the proposed environment. Except where explicitly stated, no white space characters are allowed within the text matched by a single syntax rule for a token.

<u>Character classes:</u>

The character classes used in knowledge-based expressions are exactly the same as for the Information Container API - see page 74, production rules (1)-(8).

<u>Keywords:</u>

In the below list, the production rules (10), (11), (13), (15), (17), (22), (25)-(29), (31)-(33), (35)-(41) and (44) define *reserved words*, whereas all other keywords are positional, i.e. they are meaningful as such only in their respectively defined context.

However, even though the use of these positional keywords for symbol and variable names does not violate the given syntax rules, it is better to avoid them for such purposes. Instead, where necessary, the alternative option given by rule (59) further below should be used.

```
(9)  A        = 'A'.
(10) ALL      = 'ALL'.
(11) AND      = 'AND'.
(12) AT       = 'AT'.
(13) BOOLEAN  = 'BOOLEAN'.
(14) CLASS    = 'CLASS'.
(15) DO       = 'DO'.
(16) EXACTLY  = 'EXACTLY'.
(17) FALSE    = 'false' |
                'FALSE'.
(18) FOR      = 'FOR'.
(19) HAS      = 'HAS'.
(20) IN       = 'IN'.
(21) INSTANCE = 'INSTANCE'.
(22) INTEGER  = 'INTEGER'.
(23) IS       = 'IS'.
(24) LEAST    = 'LEAST'.
(25) LISP     = 'LISP'.
(26) LISTOF   = 'LISTOF'.
(27) LOGICAL  = 'LOGICAL'.
(28) MAX      = 'MAX'.
(29) MIN      = 'MIN'.
(30) MOST     = 'MOST'.
(31) NONE     = 'NONE'.
(32) NOT      = 'NOT'.
(33) NUMBER   = 'NUMBER'.
(34) OF       = 'OF'.
(35) ONEOF    = 'ONEOF'.
(36) OR       = 'OR'.
(37) REAL     = 'REAL'.
(38) SETOF    = 'SETOF'.
(39) STRING   = 'STRING'.
(40) THIS     = 'THIS'.
(41) TRUE     = 'true' | 'TRUE'.
(42) TYPE     = 'TYPE'.
(43) VALUE    = 'VALUE'.
(44) UNKNOWN  = 'unknown' |
                'UNKNOWN'.
```

Lexical elements:

As can be easily seen, most of the lexical elements detailed below are very similar or exactly the same as in the Information Container specification shown in section 4.2. The definition of **variable** introduces the possibility for using *free variables* within knowledge-based expressions. Such variables are bound at execution time to the result values of knowledge-base terms or templates. *Operators* (**template_op**, **term_op**, **rel_op**) provide for greater flexibility and can be used to construct complex expressions by combining the basic templates defined in rules (68)-(76) in many different ways.

```
(45) boolean      = TRUE | FALSE .
(46) logical      = TRUE | FALSE | UNKNOWN .
(47) longint      = [ sign ] { digit }+ .
(48) natural_number  =  { digit }+ .
(49) real         = [ sign ] { digit }+ '.' { digit }*
                    [ { 'E' | 'e' } [ sign ] { digit }+ ] .
(50) string       = '"' { non_q_char | quote_char | ' ' }* '"' .
(51) simpleType   = BOOLEAN | INTEGER | LOGICAL | NUMBER | REAL |
                    STRING .
(52) symbol       = letter { letter | digit | '_' }* .
(53) refID        = modelID | objID .
(54) modelID      = ID .
(55) objID        = [ modelID ':' ] ID .
(56) ID           = symbol [ '.' { symbol | longint } [ version ]] .
(57) version      = ';' longint .
(58) variable     = '?' symbol .
```

```
(59) keyword     = [ '_' ] symbol .
(60) template_op = ALL | NONE | NOT | AND | OR .
(61) term_op     = ONEOF | LISTOF | SETOF | MAX | MIN .
(62) rel_op      = '=' | '<' | '>' | '<=' | '>=' | '<>' | ':=:' .
(63) sign        = '+' | '-' .
```

Grammar rules

The following high-level production rules specify how the above tokens can be combined to form valid knowledge-based expressions. In order to avoid ambiguity, `whitespace` characters may be used as separators between the individual tokens.

```
(64) searchExpression = '(' FOR { ALL | { variable }+ } DO
                            KB_Template ')' .
(65) KB_Template      = basicTemplate | complexTemplate |
                        constraintTemplate .
(66) basicTemplate    = subclassTemplate | instanceTemplate |
                        attrExistenceTemplate |
                        attrValueTemplate | attrTypeTemplate |
                        maxCardinalityTemplate |
                        minCardinalityTemplate |
                        eqCardinalityTemplate .
(67) complexTemplate  = '(' template_op { KB_Template }+ ')' .
(68) subclassTemplate = '(' subclass IS A class ')' .
(69) instanceTemplate = '(' instance IS [ INSTANCE ] OF class ')' |
                        '(' instance IS IN [ CLASS ] class ')' .
(70) attrExistenceTemplate  = '(' attr OF instance ')' .
(71) attrValueTemplate      = '(' attr OF instance
                                  HAS VALUE KB_Term ')' .
(72) attrTypeTemplate       = '(' attr OF instance
                                  HAS TYPE v_Type ')' .
(73) maxCardinalityTemplate = '(' instance HAS AT MOST
                                  natural_number  attr ')' .
(74) minCardinalityTemplate = '(' instance HAS AT LEAST
                                  natural_number  attr ')' .
(75) eqCardinalityTemplate  = '(' instance HAS EXACTLY
                                  natural_number  attr ')' .
(76) constraintTemplate     = '(' KB_Term rel_op KB_Term ')' .
(77) KB_Term          = valueSelect | variable |
                        '(' term_op { KB_Term }+ ')' .
(78) valueSelect      = literal | KB_Template | lisp_expr |
                        refID | THIS .
(79) v_Type           = symbol | simpleType .
(80) class            = symbol | variable | THIS .
(81) subclass         = symbol | variable | THIS .
(82) instance         = refID | variable | THIS .
(83) attr             = symbol | variable .
(84) literal          = boolean | logical | longint | real | string |
                        symbol .
(85) lisp_expr        = '(' LISP lisp_form ')' .
```

Informal propositions

Most of the propositions regarding the Information Container specification detailed in section 4.2 hold true here as well. The main differences are in the treatment of some of the EXPRESS data types, in the use of LISP forms, and in statements (7) to (10) below, concerning the use of `searchExpression`, `KB_Template` and `KB_Term`.

1) The **mapping of entities defined in EXPRESS data models** is done as follows:

   – Class names are represented as symbols;

   – Enumeration items are also represented as symbols; this does not contradict to the EXPRESS syntax because class names and enumeration items are not allowed to have identical names within the same data model;

   – User-defined data types are represented with their underlying base types;

   – The `SELECT` type is always represented with one of its actual underlying data types. For example, when using the `attrTypeTemplate`

   ```
   (attr OF instance HAS TYPE v_Type)
   ```

   with `v_Type` corresponding to a `SELECT`, just one of its possible underlying types may be chosen. If more than one of the underlying types are needed, a complex template of the form

   ```
   (OR (attr OF instance HAS TYPE type1) ...
       (attr OF instance HAS TYPE typek))
   ```

   should be used for the desired subset of underlying types defined by the `SELECT`;

   – Lists and Arrays are represented with the help of the term operator `LISTOF`, whereas sets and bags are represented with the help of the `SETOF` operator, e.g.

   `(LISTOF 1 2 3 5 7 11)` or
   `(SETOF IfcBeam.1170 IfcBeam.34993 IfcBeam.19288)`

   – References to object instances are represented simply by their `refID`'s;

   – `INTEGER` is represented as `longint`, and `NUMBER` is represented according to the respective actual value (`longint` or `real`);

   – `BOOLEAN`, `LOGICAL`, `REAL` and `STRING` are represented according to rules (45), (46), (49) and (50) respectively.

2) **`longint`** values may contain as many digits as allowable in the internal representation of long integer numbers, i.e. a `longint` should represent a valid integer number in the range $[-2^{64} ; 2^{64} -1]$.

3) **`real`** values are interpreted internally as double precision floating point numbers.

4) ***lisp_form***, referenced in rule (85), is an external construct which stands as a substitute for any special rules that might be needed for the specification of arithmetic or relational expressions. A *lisp_form* can be a valid Common LISP top-level form or a LAMBDA expression as defined in (Steele 1990), but should always return the value type required at the place where it is used. It may include:

   – local variable bindings introduced by the Common LISP LET construct;

   – free variables as defined in rule (58) above;

   – any of the standard Common LISP functions, as far as they do not refer to additional user-defined routines.

5) An internal representation for each **keyword** in the form '**`_symbol`**', as given by rule (59), is provided for the cases when a keyword exists also as a name of some element in an EXPRESS schema. For example, if the name “`INSTANCE`“ exists in a data model, '**`_INSTANCE`**', instead of '**`INSTANCE`**' should be used if necessary to avoid ambiguity.

6) **Upper and lower case** in all names defined in EXPRESS schemas should be preserved because a client implementation written in a language like C++ or Java may rely on such case-sensitive names. The syntax specification itself does not impose explicitly any case-sensitivity rules.

7) **`KB_Template`**, as defined in rule (65), can be used not only in queries, but also to make new assertions for a given product model. As a general rule, if a `KB_Template` does not include a free variable, it represents a *fact* which is added to the product model as new data, provided that it does not already exist and does not contradict to the definitions in the underlying model schema. On the contrary, if a free variable is specified for a class, subclass, instance or attribute, the `KB_Template` expresses a query w. r. t. this free variable. For example, the template
```
   (UserDefinedType OF IfcBeam.123 HAS VALUE "Simple Beam")
```
may fill the attribute `UserDefinedType` of instance `IfcBeam.123` with the value `"Continuous Beam"`, if no value of this attribute yet exists, whereas the template
```
   (UserDefinedType OF ?B HAS VALUE "Simple Beam")
```
will bind `?B` to a list of all instances with `UserDefinedType = "Simple Beam"`.

8) In contrast to templates, **`searchExpression`** is used basically for query purposes. A search expression returns a list of the values of the free variables defined in its `FOR`-clause or, when the form `'FOR ALL'` is used, of all free variables introduced in the associated `KB_Template`.

For example, the expression
```
   (FOR ?A DO
           (AND (?B IS INSTANCE OF IfcBeam)
                (UserDefinedType OF ?B HAS VALUE ?A))
    )
```
will return a list of all values of the attribute `UserDefinedType` of all existing “beam” instances, whereas the expression:
```
   (FOR ALL DO
            (AND (?B IS INSTANCE OF IfcBeam)
                 (UserDefinedType OF ?B HAS VALUE ?A))
    )
```
will return two parallel lists – one with the values of `UserDefinedType`,
and one with the `refID`’s of the “beam” instances themselves.

It is an error if a free variable is specified in the `FOR`-clause of a search expression, but does not appear in the associated template(s).

9) **Assertions through knowledge-based templates** are intended mainly for rules that would be implemented on the server.

Assertions of new facts by a client application can be done only as “side effects” of a query, e.g.:

```
(FOR ALL DO
         (AND (UserDefinedType OF IfcBeam.123
                   HAS VALUE "Continuous Beam")
              (calcBeamSectionArea OF IfcBeam
                   HAS VALUE ?A))
 )
```

Here the first template asserts a fact, whereas the second represents the actual query.

10) At last, the keyword **THIS** is foreseen for embedding of search expressions in Information Container based requests and in mapping specifications.
In the first case, **THIS** is automatically bound to the object that is issuing the request and can be used in the place of `class`, `subclass` or `instance` in a `KB_Template`; when used as `class` or `subclass` it will be substituted with the name of the class of the actual instance, and when used as `instance` - with its respective `refID`.
In the second case, **THIS** is bound to the currently processed instance of a source class in an inter-class mapping definition and can be used in any mapping construct where a `simpleTemplate` is referenced[*)].

4.7.3 Examples

The developed representation approach allows to perform knowledge-based search directly, i.e. through the user interface of the project data server, or to embed search expressions in any server routine, in Information Container based requests, in rules and in mapping specifications. The small examples given below demonstrate some of these capabilities.

<u>Example 1:</u> *adapted from ToCEE*

In order to retrieve the object references of all simple beams contained in a structural domain model, i.e. all instances of *IfcBeam* which are defined with the *GenericType* `SIMPLEBEAM`, the following search expression can be specified:

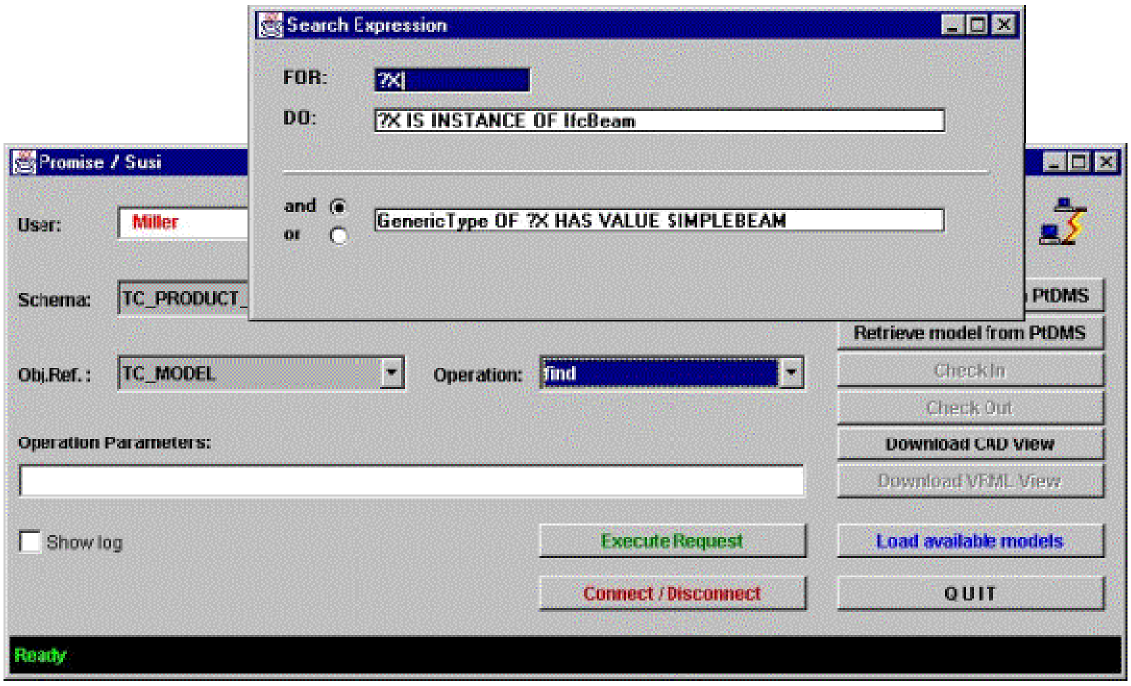
```
(FOR ?X DO (AND (?X IS INSTANCE OF IfcBeam)
                (GenericType OF ?X HAS VALUE SIMPLEBEAM))
 )
```

Use of the above search expression in the server operation ‘find’:

```
request (OID:"TC_MODEL.Struct_Model_1" meth:"find"
         inParams:
          ic(searchExpr:
             "(FOR ?X DO
                   (AND (?X IS INSTANCE OF IfcBeam)
                        (GenericType OF ?X HAS VALUE SIMPLEBEAM)))"
             )
         )
```

*) This issue is discussed in more detail in chapter 6 presenting the CSML mapping language developed in the thesis.

Use of the above search expression in the prototyped sample project data client:


Search Expression

FOR: ?X

DO: ?X IS INSTANCE OF IfcBeam

[✓ and] or

GenericType OF ?X HAS VALUE SIMPLEBEAM

Promise / Susi

User: Miller

Schema: TC_PRODUCT_

Obj.Ref.: TC_MODEL

Operation: find

Operation Parameters:

Show log

Ready

Fig. 4.9: Screenshot of a prototyped client showing the use of search expressions

Comment:

This fairly simple example demonstrates a common filtering case (similar to a projection in database terminology). It is included here because (1) such SQL-like constructions are assumed to be useful in quite a few situations, (2) it shows the basic procedure for embedding search expressions in server requests, and (3) it gives an impression how a GUI can be constructed to enable interactive user input of knowledge-based queries.

Example 2: *adapted from ToCEE*

In order to find out if there exist rooms in a building with area > 30 sq. m (a query actually used by the facility management system in the prototype ToCEE environment), the following request can be specified:

```
request (OID:"IfcSpace" meth:"find"
         inParams:
           ic(searchExpr:
               "(FOR ?S DO
                  (AND (?S = THIS)
                       (calcTotalArea OF ?S HAS VALUE ?A)
                       (?A > 30.0)))"
              forModel:oRef(TC_MODEL.Arch_Model_X)))
```

Comment:

In this example the keyword `THIS` represents any instance of *IfcSpace*. The free variable `?A` is used within the search expression only as intermediate storage for values passed from the second to the third template. It is not returned by the find operation. If the values of `?A` are also needed, the `FOR` clause should be modified to `(FOR ?S ?A DO ... )`.

Example 3: *adapted from (Scherer & Katranuschkov, 1999)*

This example shows how knowledge-base templates can be used in rules. The simple rule presented below has been experimentally prototyped to examine the usefulness of automatic monitoring of the states of the project models on the project data server, in conjunction with the issues discussed in section 4.6.

Each time when a modified version of an instantiated model (`M1`) is *checked in* at the server and its status is set to *check-in* (`CI`), e.g. through a respective remote method call from a client application, the rule will be automatically triggered, allowing a separate process to determine all changes done to the model (compared to any other “active” instantiated model version with the same `projectID`), and notify the affected client(s) and/or user(s).

```
IF (AND
     (modelState OF M1 HAS VALUE CI)          ;; trigger (CI=CheckIn)
      (?n IS INSTANCE OF MODEL)
      (modelState OF ?n HAS VALUE ?ns)
      ((projectID OF ?n) = (projectID OF M1))
      (NOT (?ns = (ONEOF W COM RC)))          ;; not Write/Commit/RtC
   )
THEN
   ;; set temporary exclusive lock for a match operation
   (modelState OF M1 HAS VALUE EXCLUSIVE-TEMP-LOCK)
   (modelState OF ?n HAS VALUE EXCLUSIVE-TEMP-LOCK)
   ;; execute the matching operation using a wrapper
   ;; for the resp. public operation defined in EXPRESS-C
   (LISP
    (exec.operation 'match'(OID:M1 inParams:ic(refModel:oRef(?n))))
    )
   ;; restore the model locks
   (modelState OF M1 HAS VALUE CI)
   (modelState OF ?n HAS VALUE ?ns)
```

<u>Comment</u>:

Five templates are used in the rule premise here, specifying the necessary conditions for the rule to fire. In addition, one basic template **`(modelState ... )`** is used four times in the rule conclusion to assert the new fact, illustrating how templates can be used not only for queries, but also for manipulating the data in the project data repository.

`Exec.operation` is a generic method implemented on the project data server to enable local calls of the server operations; in this case it invokes the operation **`match`** used to compare two versions of a model[*)].

[*)] The model matching method is outlined in chapter 8; the specification of the 'match' operation in its general, Information Container based format is given in appendix V.

Example 4: *adapted from COMBI*

Assuming that the axes of building elements are always defined by topological edge entities with vertex points specified in a global coordinate system, the following expression can be used to collect all building elements that lie on a given plane z = 3.60 m.

```
(FOR ?X DO
  (AND (axis OF ?X HAS VALUE ?A)
       (coordinates OF (point OF (edge_start OF ?A)) HAS VALUE ?P1)
       (coordinates OF (point OF (edge_end OF ?A)) HAS VALUE ?P2)
       ((LISP (THIRD ?P1)) = (LISP (THIRD ?P2)))))
```

Comment:

This example shows how entities can be grouped in sets with the help of knowledge-base templates, even if such grouping is not explicitly defined in the underlying model schema. The third and the fourth line demonstrate how basic templates can be combined into more complex expressions without the need to introduce intermediate variables. The last line shows how LISP expressions can be used; in this case - to access the third element in each list of coordinates bound to `?P1` and `?P2`.

Example 5: *use of a search expression with IFC model data (IAI 1999c)*

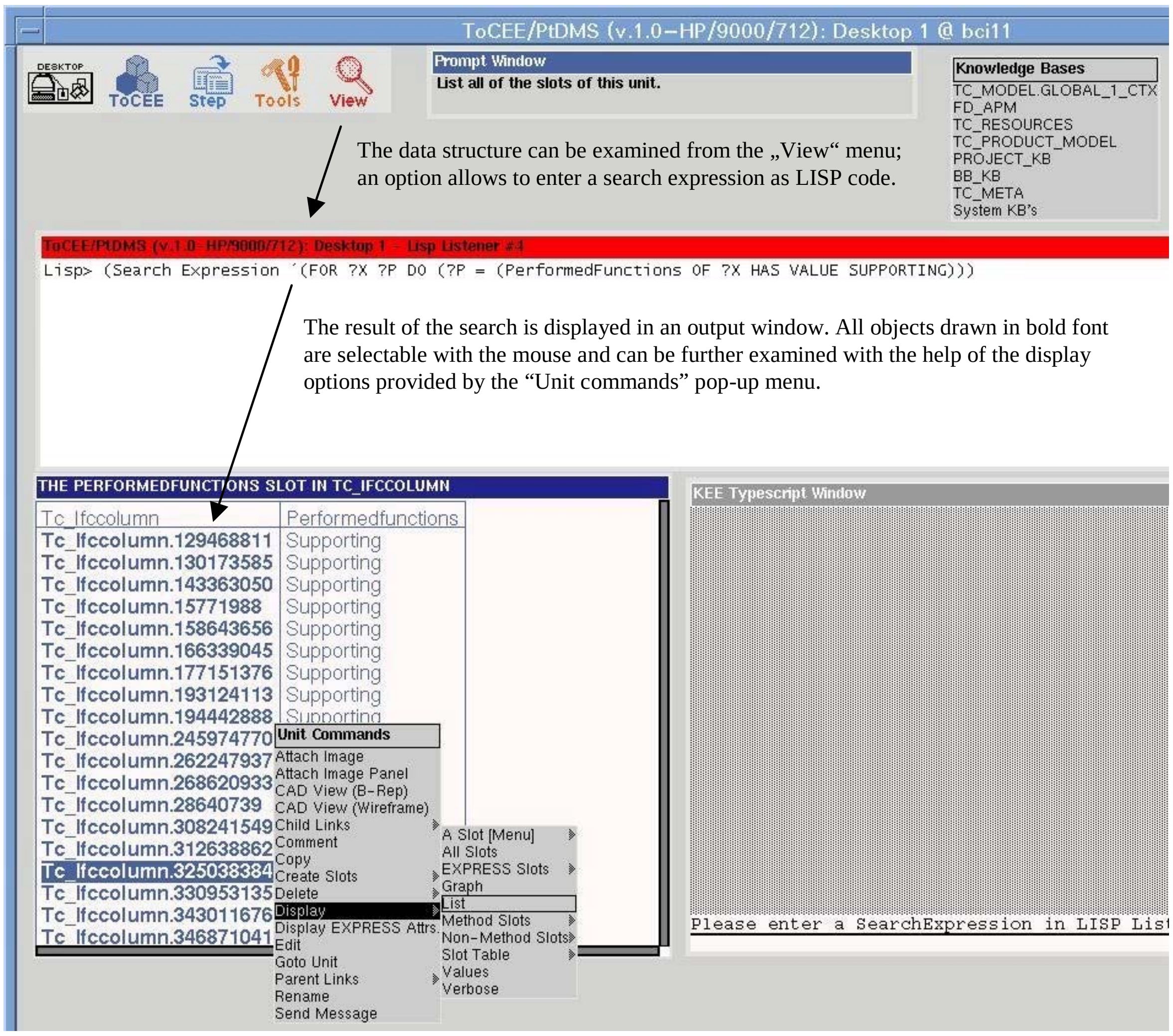
In order to retrieve all load-bearing elements contained in an IFC-based building model, regardless of whether they are beams, columns, walls or slabs, the following search expression can be used:

```
(FOR ?X ?P DO (?P = (PerformedFunctions OF ?X HAS VALUE SUPPORTING)))
```

Comment:

The purpose of this search expression is in principle very similar to that used in example 5. It is included here basically to demonstrate that different product models can be processed in much the same way, as well as to give a hint for the possible practical benefits of the use of search expressions in IFC-based applications. In the given case, the needed result is achieved with one single line of code which can be embedded as a string in a 'find' request to the project data server, in a language and platform independent way. In a more traditional approach, an iteration over the instances of all object classes, involving several SDAI-like requests to the project data server and a notable communication overhead, would be needed to obtain the same result.

Fig. 4.10 on the next page demonstrates the interactive use of the search expression with the GUI of the project data server. This can be helpful for examining the model data structures directly on the server by an authorised person, such as a system administrator, an information manager or the project manager. Such activities are facilitated by the uniformity of the representation of operations (with the unavoidable differences from specific language bindings).



ToCEE/PtDMS (v.1.0–HP/9000/712): Desktop 1 @ bci11

Prompt Window
List all of the slots of this unit.

Knowledge Bases
TC_MODEL.GLOBAL_1_CTX
FD_APM
TC_RESOURCES
TC_PRODUCT_MODEL
PROJECT_KB
BB_KB
TC_META
System KB's

The data structure can be examined from the „View“ menu; an option allows to enter a search expression as LISP code.

ToCEE/PtDMS (v.1.0–HP/9000/712): Desktop 1 – Lisp Listener #4

```
Lisp> (Search Expression ´(FOR ?X ?P DO (?P = (PerformedFunctions OF ?X HAS VALUE SUPPORTING))
```

The result of the search is displayed in an output window. All objects drawn in bold font are selectable with the mouse and can be further examined with the help of the display options provided by the “Unit commands” pop-up menu.

THE PERFORMEDFUNCTIONS SLOT IN TC_IFCCOLUMN

Tc_Ifccolumn	Performedfunctions
Tc_Ifccolumn.129468811	Supporting
Tc_Ifccolumn.130173585	Supporting
Tc_Ifccolumn.143363050	Supporting
Tc_Ifccolumn.15771988	Supporting
Tc_Ifccolumn.158643656	Supporting
Tc_Ifccolumn.166339045	Supporting
Tc_Ifccolumn.177151376	Supporting
Tc_Ifccolumn.193124113	Supporting
Tc_Ifccolumn.194442888	Supporting
Tc_Ifccolumn.245974770	
Tc_Ifccolumn.262247937	
Tc_Ifccolumn.268620933	
Tc_Ifccolumn.28640739	
Tc_Ifccolumn.308241549	
Tc_Ifccolumn.312638862	
Tc_Ifccolumn.325038384	
Tc_Ifccolumn.330953135	
Tc_Ifccolumn.343011676	
Tc_Ifccolumn.346871041	

KEE Typescript Window

```
Please enter a SearchExpression in LISP List
```

Fig. 4.10: Use of a search expression with the GUI of the project data server

4.8 Discussion

Whilst in the basic client/server model, shown on fig. 4.1 at the beginning of this chapter, the only actual concern is to establish an appropriate communication protocol for client/server interaction, in a distributed, highly flexible CEE system for building design there are many challenging problems that need to be resolved. Major problems provide:

– the heterogeneity of the system components,
– the use of different communication paradigms,
– the use of different data exchange formats and different data transmission and data transaction modes (synchronous/asynchronous, batch/interactive, long/short read/write transactions),
– the possibly different data representation paradigms of the component applications, and
– the specific concurrent engineering requirement for simultaneous access to shared data, along with independent local processing of this data.

Many of these problems have been intensively studied in computer science research. However, while a number of principal solutions for different systemic interoperability aspects exist, none of these solutions is capable to cover the full range of requirements to a CEE system.

For example, the CORBA model (Orfali et al. 1997) provides a comprehensive solution for a distributed object system, enabling platform and language independent remote method invocation, but it does not define which methods of which objects are appropriate to be used remotely. Similar, yet different solutions are provided by Java RMI and the Microsoft DCOM model, but they also do not give any hints as to what should be the content and the functionality of the distributed objects in the system. In fact, while each of these approaches can be helpful for CEE, altogether they increase the complexity of the system as they all need to be simultaneously supported. Other communication methods, such as "raw" TCP/IP, HTTP and FTP also bring more problems than remedy.

Much the same situation exists with respect to data exchange. STEP provides both a standard data access interface, SDAI (ISO 10303-22 1998), and a file exchange format (ISO 10303-21 1994), but does not go on to define any objects and methods to utilise these specifications. IFC uses the same approach, but also does not proceed to define implementation methods w.r.t. a run-time system. Other formats that might need to be supported, such as VRML or DWF, only make the situation worse because of the different representation paradigms they use.

Similar problems appear with respect to data sharing. For example, SDAI defines the concepts of sessions and transactions, but does not deal with problems related to long transactions where a large portion of a model is checked out for local processing for a longer period of time. Other solutions known from database research can provide help in that respect, but they are mainly developed for the relational data model which is not directly compatible to EXPRESS (cf. Dadam 1996).

Thus, the main systemic interoperability problem lies in fact in the necessity to consider all component aspects in their entirety, as part of a coherent overall solution.

This problem can be largely overcome with the developed approach of inter-linked specifications and communication and data management methods presented in this chapter, as:

1) It provides a consistent realisation of the idea of Information Containers outlined in chapter 3 and suggests an efficient, simple to implement method for the use of various communication paradigms for the execution of different project data related operations.
2) Through the definition of the concept of *MODEL* objects, capabilities for data exchange both on "macro" level (with data exchange files, such as STEP physical files) and on "micro" level (with SDAI-like operations) are provided in a uniform way.
3) The developed "light-weight" formalism for the specification of knowledge-based queries enables the use of advanced server features by traditional procedural or object-oriented applications without any reasoning capabilities. This pragmatic approach provides less functional features than comprehensive solutions such as ACL/KIF (Khedro et al. 1994), or the knowledge-based extensions integrated externally in the ATLAS project (Poyet et al. 1994b), but it allows easy integration of many typical engineering tools with limited cognitive capabilities. Thus, in the performed case studies, and in the frames of the ToCEE project, a tool for foundation design, a structural analysis system and a facility management system could utilise the server functionality with minimal development efforts (Scherer 2000).

4) By a suggested XML externalisation of the high-level data structures (Information Containers, Requests, Responses), presented in Appendix III at the end of the thesis, pure WWW-based communication and data exchange can be accomplished in addition to the methods of discourse on application level, enabled by the generalised communication model. In this way, fully transparent, independent of the network protocols client-server interaction can be realised.

As a whole, the approach is easily extensible, because on the basis of a lean meta model, a standard kernel model (IFC) and a set of basic operations, various project data services can be realised, as suggested in section 4.5. The aligned specifications for Information Containers, communication data structures, object-oriented operations and knowledge-based extensions are fully implementable and have been verified in the prototyped environment described in chapter 8. However, the main advantage of the developed approach is not so much in the proposed specifications, but in the overall consistent treatment of the full spectrum of systemic interoperability problems listed at the beginning of this section. The alternative XML based syntax for Information Containers, Requests and Responses – developed in only a few days on the basis of the conceptual Information Container and Communication model schemas – underpins the value of the suggested principal solution concept. Another contribution provide the methods enabling long transactions which build the basis for the tackling of concurrent building design processes, as outlined in section 3.7.

However, a final solution even on this basic interoperability level is not yet achieved. Further work can be envisaged at most component level, as suggested below.

1) *Information Containers*

   On the level of Information Containers, an extension of the XML externalisation is worth considering. It would be interesting to examine the possibility to use the proposed new *XML Schema* (Fallside et al. 2000) instead of the currently suggested XML DTD to provide better coverage of the Information Container semantics. Useful ideas for further improvements can be obtained also from the work on a XML specification for EXPRESS-driven data undertaken in ISO STEP (ISO 10303-28 1999). A mapping of Information Containers to other data formats, such as XHTML, is another topic that can be studied.

2) *Communication model*

   On the level of the communication model, a multi-server system and a server interoperability protocol can be useful extensions worth to be considered.

3) *Object-oriented operations*

   The definition of a well-defined set of operations is not yet available. Many useful hints can be taken from the area of database research, but as the problem domain and the targeted functionality are different, a deeper analysis of the information processes for the identification of a comprehensive set of elementary operations needs to be performed.

4) *Transactions and model states*

   In the developed approach, problems related to incremental updates are not considered. However, it would be desirable to define formalised methods not only to store incremental updates, but to define and exchange update information on the basis of model-level operations. Such a feature can be very useful, as many design processes only change a portion of the checked out data. Fig. 4.11 below presents the basic idea, but there are many detail problems that have to be solved in order to provide the suggested functionality.

Second, while model states and compatibility of operations have been basically identified (see section 4.6), additional efforts are needed w.r.t. data recovery and data coordination.

Finally, much further research is needed w.r.t. the features of long transactions. Here there is little work done even in the area of basic database research. For example, concepts like semantic and application-specific synchronisation methods can be imagined, and better support for version control needs yet to be provided.

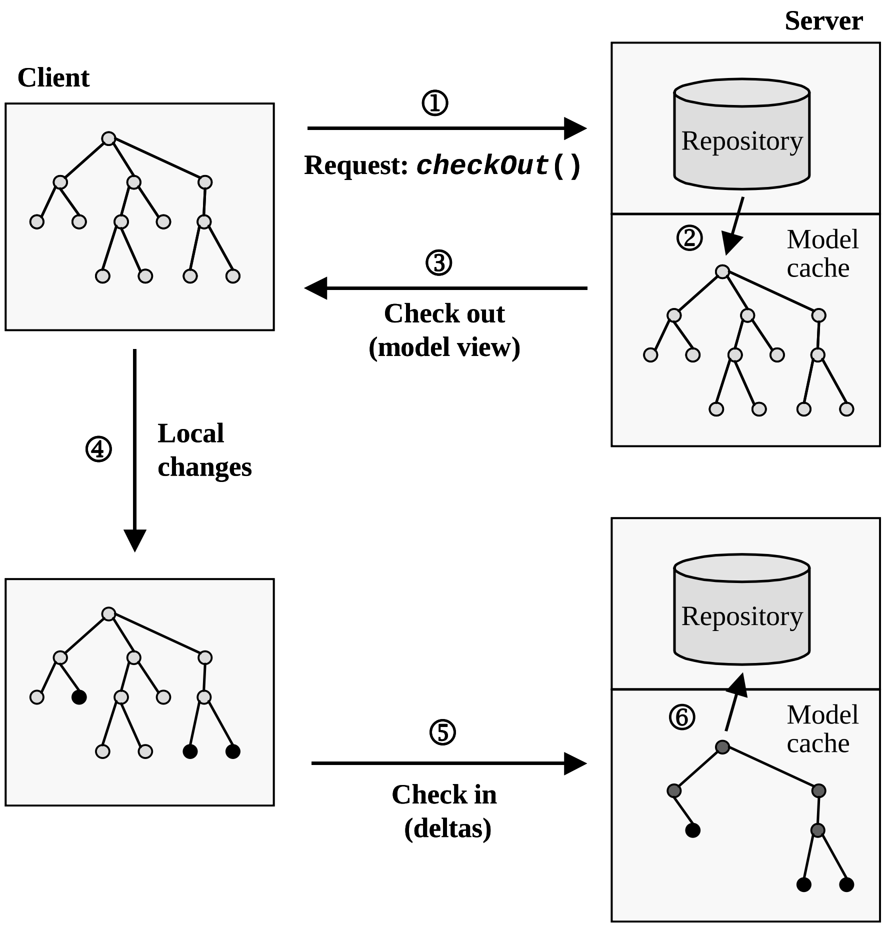


Fig. 4.11: Principal schema of incremental model data updates by long transactions

The numbers in the figure indicate the sequence of operations, as follows:
(1) request to check out a model view by the client
(2) caching the model for further processing by the server
(3) performing of the actual check out
(4) local changes to the model by the client application (shown by black dots)
(5) check in only of the changed data (deltas)
(6) updating of the server repository on the basis of the received change data.

Chapter 5: Semantic Interoperability

What is in a name? That which we call a rose
By any other name would smell as sweet.
– William Shakespeare, Romeo and Julia

Semantic interoperability presents the second, and most challenging, category of interoperability problems that have to be tackled in a CEE system. It is insufficiently investigated, and its theoretical foundations are not clearly identified. Because of the lack of high-level generic solutions, in most existing integration environments the problems of semantic interoperability are only weakly supported by dedicated software tools, leaving most of the work to ad hoc coded interfaces. This reduces considerably the flexibility, the extensibility and the maintainability of the overall environment.

In this and the next two chapters a novel approach to the solution of the basic semantic interoperability problems relevant to CEE will be presented. The focus of this chapter is on the overall conceptual issues and the state of research and development w.r.t. semantic interoperability, whereas the next two chapters cover specific technical aspects of the proposed new approach.

As a starting point, the problem domain and three alternative solution strategies are outlined. It is shown that *model harmonisation* à la STEP and IFC can considerably reduce the complexity of the task, but is not likely to be a universal solution for all kinds of problems that have to be dealt with. *Model integration*, known from database research, provides useful background knowledge, but is also insufficient for the targeted problem domain. Therefore, the different types of *mappings* needed to describe the correspondences between non harmonised or only partially harmonised conceptual models are analysed in detail, and a set of *mapping patterns* conceived as basic building blocks for the solution of most practical mapping problems is proposed. Existing mapping approaches are also examined, and yet missing features are identified. At the end, a *new model mapping approach* is suggested, and the aspects that distinguish it from other known approaches are briefly discussed.

5.1 Basic Concepts

From the standpoint of application developers and end-users of a CEE system semantic interoperability is a problem of *understanding*.

However, from the standpoint of the system developer it is, again, a problem of appropriate *conceptualisation*. In short it can be defined as the ability of conceptual model schemas to share common concepts which can then be used for the actual exchange and sharing of information stored in respective repositories according to these conceptual schemas.

Ideally, in a fully interoperable environment all these schemas could be viewed as a single logical system, as if they were parts of one single global schema.

In practice, as illustrated on fig. 5.1, it does not yet work like that.

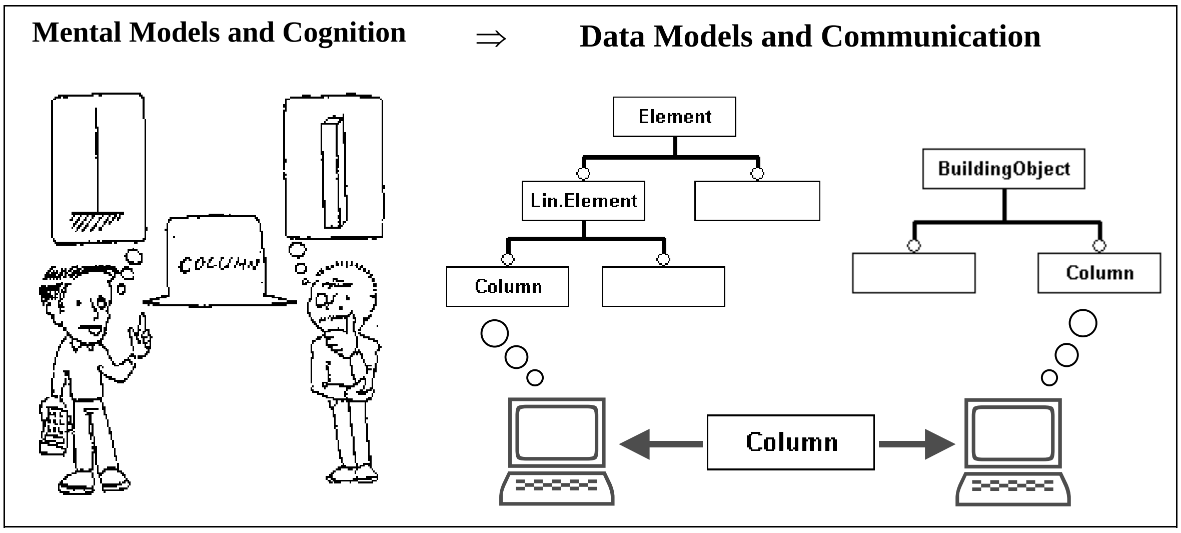


Fig. 5.1: The general semantic interoperability problem on the example of the object "Column" / adapted from (Junge et al. 1995b) /

First, in the heterogeneous, fragmented world of CAE/CIC it is unrealistic to assume that all component models (architectural, structural, HVAC etc.) can a priori be tightly harmonised and integrated. It is not easy, if at all possible, to derive a common model for all applications needed in a construction project, and even on a higher level of abstraction (e.g. for project management) there do not yet exist such unified models. The major problem in that respect is that the scopes of sending and receiving applications are not likely to be known in advance, and – due to the fuzzy boundaries of AEC and the wide range of technologies involved – the information needs in general are unpredictable (Hannus et al. 1995b).

Second, even if the specification of a common (super)model was realised in some way, it would still be necessary to derive discipline-specific views that correspond to the real-world abstractions used by the different players in a construction project (architects, structural engineers, cost estimators …). In order to achieve a wide cooperation in CEE it should be possible to work on such discipline-specific models individually and in parallel, but at the same time the information contained in each individual model should be available "when needed and as needed" (Scherer 1998a; Turk et al. 2000).

With some imagination, it is not difficult to recognise these aspects of semantic interoperability even on the simple example given in fig. 5.1 above. More elaborate scenarios, as the one presented later on in chapter 8, only emphasise the fact that semantic interoperability is in fact a *model transformation problem for a distributed non homogeneous model world.*

In a concurrent engineering environment such model transformations may be needed e.g. for:

- data exchange between major CAD applications;
- generation of views for presentation, documentation, pre- and post-processing purposes;
- detection of conflicts between different design solutions;
- coordinated change management including propagation of the changes made in one domain to all other affected domain-specific representations of the same design objects;
- extracting of management information for overall control and monitoring etc.

In order to support such tasks, the design of an interoperable concurrent engineering environment must incorporate:

1) an **appropriate strategy** to "assemble the pieces of the puzzle", that fits into the overall architecture of the environment;
2) **description methods** to represent the inter-model correspondences that should ensure consistent data transformations across model views and application platforms;
3) **implementation methods** to enable the realisation of the developed formal concepts in a running software system.

An approach that combines all these design aspects in a coherent way will be presented at the end of this chapter. However, before that the separate components of the problem have to be investigated. It is necessary to outline the problem domain, to examine possible solution strategies and related approaches, and to analyse all those issues that may influence a model transformation task.

5.2 The Problem Domain

On the one hand, the subject of semantic interoperability in each specific environment are, obviously, the different representations of the project data in the models within the scope of that environment. Thus, it seems that the domain of the problem would differ from case to case, but will be well outlined by the set of data definitions in the conceptual models under consideration.

Unfortunately, this "revelation" does not bring much. Indeed, any two models can easily be connected through a more or less sophisticated interface derived from the examination of the relevant data structures, but for N models nearly $N^2/2$ interfaces will have to be written, and even with a shared central model at least N interfaces will still be needed. Though practiced with success in many integrated environments of limited scope and pre-defined architectures (see e.g. Pohl et al. 1992; Augenbroe 1995a; Watson & Crowley 1995), this approach is not satisfactory for any more complex real-world environments. Thus, a more general characterisation of the problem has to be found.

On the other hand, taking in consideration that any model has certain generic features, independent of the particular data definitions and content, it should be possible to outline the problem domain not in terms of the concrete modelling objects, but through the principal relationships within and between the models.

By looking at the problem from that perspective a similarity to the research in the area of database integration can be recognised. In fact, the problems related to model transformations between conceptual schemas, commonly known as *mapping problems*, have been intensively studied in the last years, as part of the pre-integration phase in several proposed multidatabase integration approaches (Sheth & Larson 1990; Spaccapietra et al. 1992; Reddy et al. 1994; Kim 1995). Though carried out in a different area, these research efforts help to identify the "real" domain of the semantic interoperability problem in CAE/CIC, i.e. the *types of conflicts* that may appear between conceptual model schemas.

A widely acknowledged classification, drawn from (Spaccapietra et al. 1992), divides the **types of conflicts** between model schemas into four categories: *semantic conflicts*, *descriptive conflicts*, *structural conflicts* and *heterogeneity/paradigm conflicts*.

5.2.1 Semantic conflicts

Semantic conflicts appear between independently developed, or only partially harmonised schemas which represent the same or overlapping abstractions of certain real-world concepts. In such schemas it is common to find analogous classes which are, however, not absolutely identical, or whose sets of instances are not identical. For example, it is possible that a class in one schema represents a subset of the real-world objects covered by a respective class in another schema, or that there is a semantic overlapping of the instances of two classes where none of the instance sets is a subset of the other, but their intersection is not empty.

Semantic conflicts are quite common for the strongly fragmented world of CAE/CIC. Numerous examples of modelling concepts that fall in this category can be given, starting with a simple *point* (which can be defined with 2 or 3 attributes for the individual coordinates, with polar coordinates etc.) and proceeding with a long list including almost every more elaborate concept, such as *building*, *site*, *frame*, and so on. Practical examples of many such conflicts can be found by comparing the IFC Project Model (IAI 1999c) and the CIMsteel integration model (Watson & Crowley 1995), both based on the same modelling paradigm (EXPRESS) and covering overlapping design domains.

5.2.2 Descriptive conflicts

Even when the concepts defined in two model schemas do represent identical sets of real-world objects, it is possible that these objects are described by different properties (attributes). This type of conflicts can be further subdivided into naming conflicts (homonymous and synonymous attribute names), value range conflicts, default value conflicts, unit and scalability conflicts, and conflicts related to non identical integrity constraints for one and the same attribute in the two model schemas.

Descriptive conflicts are very close in meaning and importance to the semantic conflicts described above. In fact, many researchers do not consider them a separate conflict category. When two practical models are compared, as e.g. the CIMsteel model and the IFC Project Model, both categories appear mostly side by side due to the fact that different descriptions of analogous concepts are almost always associated with non identical instance sets as well.

5.2.3 Structural conflicts

Structural conflicts arise when the same real-world concept is modelled by using different modelling concepts in two different schemas, even when these schemas are developed on the basis of the same representation paradigm.

Especially in object-oriented models and in EXPRESS-based conceptual models, there are always several alternative ways to define one and the same property of a given class. It is possible to define a complex entity property by a single value attribute of set or list data type, by a fixed number of individual single valued attributes, or even by one or more references to other entities containing these attributes. For example, the developer of a structural design model may choose separate classes for building elements and loads whereas the developer of a foundation design model may define the loads on the foundation elements directly as value attributes. Another example is the presence or lack of "node" entities in these models, in the latter case replaced by simple references to the location points of the nodes - see (Katranuschkov & Scherer 1996), as well as the study presented in section 7.2. Even in a typical structural domain model, it is possible to suppress the specification of a *node* concept for the purpose of better alignment with an IFC-based architectural model, as shown in (Weise 1999) [*)].

Due to the complexity of the involved schemas, structural conflicts appear in practically every model transformation task from the AEC domain. Because of this, they have been analysed in greatest detail in all known integration and mapping approaches, and are also a prime issue in the model mapping approach developed in this thesis.

5.2.4 Heterogeneity / paradigm conflicts

Lots of problems arise also when the models to be integrated or mapped are represented by using different representation paradigms, e.g. an object-oriented model developed according to the UML modelling formalism, a STEP/EXPRESS model, a relational model etc. For this kind of conflicts there is no common recipe known, and solutions are sought individually for any pair of paradigms.

The most typical, and at the same time most challenging task here is the transformation of an object-oriented model (quite common for many of today's applications) to/from a relational data model (quite common for the database systems used in today's practice). The great difficulty is that the relational modelling paradigm offers by far less modelling concepts than the object-oriented paradigm. In (Schneider 1992) it has been shown that in the general case an EXPRESS-based data model cannot be mapped completely to a relational database because no equivalent constructs can be found for some of the EXPRESS modelling concepts[**)]. Beside this, such kinds of model transformations imply, almost inevitably, structural conflicts as well, due to the fact that the different paradigms enforce the modelling of the same real-world concepts by different modelling concepts on the detail level.

5.3 Solution Strategies

In principle, there exist three distinct strategies that can be applied to the outlined problem domain. These are: (1) *model harmonisation*, (2) *model (or schema) integration*, and (3) the proposed in this thesis strategy of *model mapping.*

Model harmonisation aims at providing a general methodology which should enable the design of a priori consistent data models, i.e. before their use in any practical environment. It is a strategy of "integration by intention", related to the process of model development itself. Typical efforts in this direction are being undertaken in ISO STEP (in general) and in IAI (specifically for the building construction sector).

In contrast, **model integration** aims at establishing a posteriori, i.e. by already existing data models, a semantically conflict-free environment with guaranteed overall consistency. It is a strategy of "integration by definition". An outstanding area of such efforts is the

---

[*)] Although the mapping to the architectural domain model will in that case be easier to achieve, this will be at the expense of the mappings eventually needed to transform the structural domain model to the data models of structural analysis applications where the separate definition of nodes is almost always explicitly required. The problem is thus only shifted, but not solved.

[**)] According to the quoted study, such "non translatable" EXPRESS constructs include local and global rules, derived attributes, functions and procedures; a complete translation is stamped as "not possible" also for complex ANDOR inheritance cases and for UNIQUE constraints.

development of federated systems for existing enterprise databases, especially where the high reliability of already existing data is of primary concern.

The essence of **model mapping**, as proposed in this study, is also in the development of methods that can be used to bring together non harmonised model data. However, instead of attempting to integrate the involved conceptual models when the environment is designed, the model mapping methods aim at enabling the semantically correct transfer of the data contained in one model into the representation required for another model *only when needed*, at run-time. Thus, by separating the integration and consistency requirements for the overall system, an "integration by demand" is hoped to be achieved.

Depending on the circumstances, these three basic strategies can be inter-mixed in some appropriate combination. However, in my opinion, due to the specific situation in the building industry and the specific requirements of concurrent engineering, the use of model mapping methods will always be needed to fill in the gaps and to glue things together.

5.4 Model Harmonisation

The harmonisation of the data models for a given technical domain already at the stage of their conceptual development, without having in mind any concrete realisation, is a primary concern in the area of *standardisation*. To support this strategy, a comprehensive methodology has been developed in the frames of the ISO STEP standard (Burkett & Yang 1995; Owen 1997), and a number of practical rules w.r.t. the procedures and stages of model development have been set up by the IAI as well (Liebich & Wix 1998; IAI 1999a, b)[*)].

Although there are some differences in the approaches, model harmonisation can generally be characterised by the following process:

1) Specification of fundamental "low-level" concepts, such as geometry, materials, units of measure etc., in "resource" schemas intended for use in any application domain
(this step is supposed to ensure that commonly needed objects like points, lines, date, time will be represented always in the same way).
2) Identification of requirements and conceptual design of each application/domain model without much attention paid to integration issues
(this step is supposed to give domain experts the freedom to define the specific information requirements of the domain without having to bother about more technical aspects).
3) Structural harmonisation of the developed preliminary version of a domain model with the resource schemas.
4) Merging of domain models into a consolidated schema
(though this is admitted to be a difficult task by most advocates of model harmonisation, the measures undertaken in steps 1-3 are anticipated to ensure the success of the effort).

[*)] Whilst both STEP and IAI name the process "model integration", the term "harmonisation" seems more appropriate here for the following reasons: (1) An integration of this kind is quite different from "model integration" as understood in computer science because it does not deal with the concrete realisation of a consistent software system, but intends to provide the prerequisites for the interoperability of any such (later developed) system; (2) The integration efforts occur at the level and with the instruments of conceptual modelling; their goal is the definition of conflict-free schemas, and not the development of methods that would enable the tackling of such conflicts when they arise.

5.4.1 Model harmonisation in ISO STEP

The STEP standard holds strictly to the development process outlined above. However, as STEP is conceived to cover the information requirements of any industry sector, and not just AEC, its methodology is respectively more complicated and requires considerable resource investments.

In general, STEP-based model harmonisation foresees the following conceptual integration stages (Burkett & Yang 1995):

1) *Intra-resource integration*: This type of integration can actually be considered finalised within STEP. Its result is a consistent set of resource schemas, the so called “Integrated Resources” (ISO 10303, parts 41 to 49). Along with a set of informal rules stating what constitutes a “good” model (generic, consistent, structurally compatible), the Integrated Resources (IR) provide the baseline of the whole harmonisation approach.
2) *Structural integration of application protocols through the integrated resources*: This type of integration is related to the prescribed procedure for the development of application protocols (APs). In order to be prepared for later integration with other APs, an application protocol is designed first as an “Application Reference Model” (ARM) which need not necessarily be aligned with any other implementable parts of the standard, but is subsequently re-designed so that the integrated resources are used at all places where respective real-world concepts have been referenced. This process is called “interpretation” to reflect the fact that the IRs, incorporated into the AP from structural standpoint, are “interpreted” by the AP which provides the specific context for their use. The resulting “Application Interpreted Model” (AIM) is supposed to be fully prepared for later harmonisation with other STEP APs. The whole procedure is strictly formalised, requires meticulous documentation, and is in general very time-consuming.
3) *Semantic integration of APs through application interpreted constructs*: This final stage of the integration process has been introduced after it was realised that the integration of the resource schemas into application protocols is not sufficient to provide the desired interoperability. The application interpreted constructs (AICs) are dedicated collections of IRs intended to serve multiple applications that would “interpret” these resources in a similar fashion (ISO/TC184/SC4/N534 1997). In this way AICs are supposed to tackle the semantic overlaps in the context of two or more APs.

Thus, if a system needs to use two or more AIM schemas, the suggested methodology is to create a new schema which includes the AIM schemas in the same way an AIM schema includes AIC schemas. This should enable the combination of multiple APs for a single implementation that will then contain sufficient information for an expanded application context while maintaining the specific constraints and identity of each individual AP to accommodate conforming data (Yang 1995).

Ideally, this strategy could lead to fully consistent models where the main model transformation problems are solved already at the outset. However, it has also some significant practical drawbacks:

– The process of model interpretation, i.e. converting an ARM to an AIM, is very elaborate; nonetheless, it does not bring the full desired effect, leaving a lot of work to the last harmonisation stage. Work of that magnitude cannot be organised easily in a fragmented industry sector like building construction. Thus, it is not surprising that at present only one “construction” AP with limited goals has been successfully finalised (AP 225 “Building Elements Using Explicit Shape Representation” – ISO 10303-225 1999).

– Discipline-specific APs for the same domain may often have different context, but many overlapping concepts (such APs are for example the drafts of AP 230 “Building Structural Frame Steelworks” and AP 228 “Building Services: Heating, Ventilation and Air Conditioning”, though both have been recently cancelled). According to the current methodology, if such APs are to be harmonised, there would be a substantial number of AICs needed which would make the whole process quite convoluted.

– At last, IRs and AICs provide actually only low-level integration support. There is no requirement for any common construction on the level of ARM, even though in some of the APs developed in STEP a high-level of harmonisation has been implicitly foreseen, as e.g. in the AP 214 “Core Data for Automotive Mechanical Design Processes” (ISO 10303-214 1997). However, such APs tend to become very large by themselves (e.g. AP 214 is documented on over 2600 pages) and it is not clear how a harmonisation with other STEP parts can ever be achieved in practice.

The above problems do not mean that the harmonisation efforts undertaken by ISO STEP have no benefits. Especially with the design of the integrated resources a clear conceptual basis for any more sophisticated models is achieved. Even though this does not bring full integration (which has actually never been more than a future vision), it nevertheless facilitates greatly the solution of many practical semantic interoperability tasks.

5.4.2 Model harmonisation in the IFC framework

The IAI adopts a more pragmatic harmonisation strategy, better suited to the available resources in the AEC sector*). Whilst the procedure is principally the same, many of the rules of the ISO STEP methodology are revised and simplified. Thus, the IFCs also make use of the STEP IRs, but their content is significantly pruned; ARM development undergoes the same design stages, but the tedious process of interpretation is completely cut off which leads to what is (unofficially) known as "implementable ARMs" (the IAI uses another terminology here, but the idea is basically the same).

However, in spite of many similarities with the STEP approach, the IFC harmonisation strategy is different in one substantial point. Taking in consideration the (positive and negative) experience from STEP and projects like ATLAS and COMBI, it introduces the concept of a *core model* as a baseline for all conceptual modelling and integration efforts.

The ideas of the core model encompass the interpretation of common requirements, the specification of common data and the development of a common *framework* (Wix 1996). The purpose of such a framework is to provide a consistent basis for the development of more specific domain data models (West & Fowler 1996).

The IAI assumes that in this way the independently developed domain models (architectural, structural, HVAC etc.) will *tend* to be consistent. Thus, it follows in effect a top-down pre-harmonisation policy whereas in ISO STEP a substantial part of the integration process is predestined to bottom-up post-harmonisation.**)

*) A short overview of the IFC methodology is presented in appendix VII, section VII.2; an overview of STEP is presented also in appendix VII, section VII.1.

**) In fact, such efforts w.r.t. the structural domain are already in progress in the Japanese Chapter of the IAI (IAI ST-2 1998), as well as in the German Chapter (not yet published).

Fig. 5.2 below shows the modular approach of the IFC architecture and provides many hints to the envisioned process of model harmonisation. However, it also gives a notion for the limits of the harmonisation approach. Thus, while the three lower layers of the architecture are fully integrated (in the sense of STEP), subtyping or using constructs deeper in the model hierarchy in a strictly pre-defined way, the domain/application layer is not. In order to achieve its integration into the overall framework, an intermediate "interoperability adapter" is introduced, along with the concept of *mapping*.

How this is going to work in practice is currently only vaguely covered in the available IFC documentation.

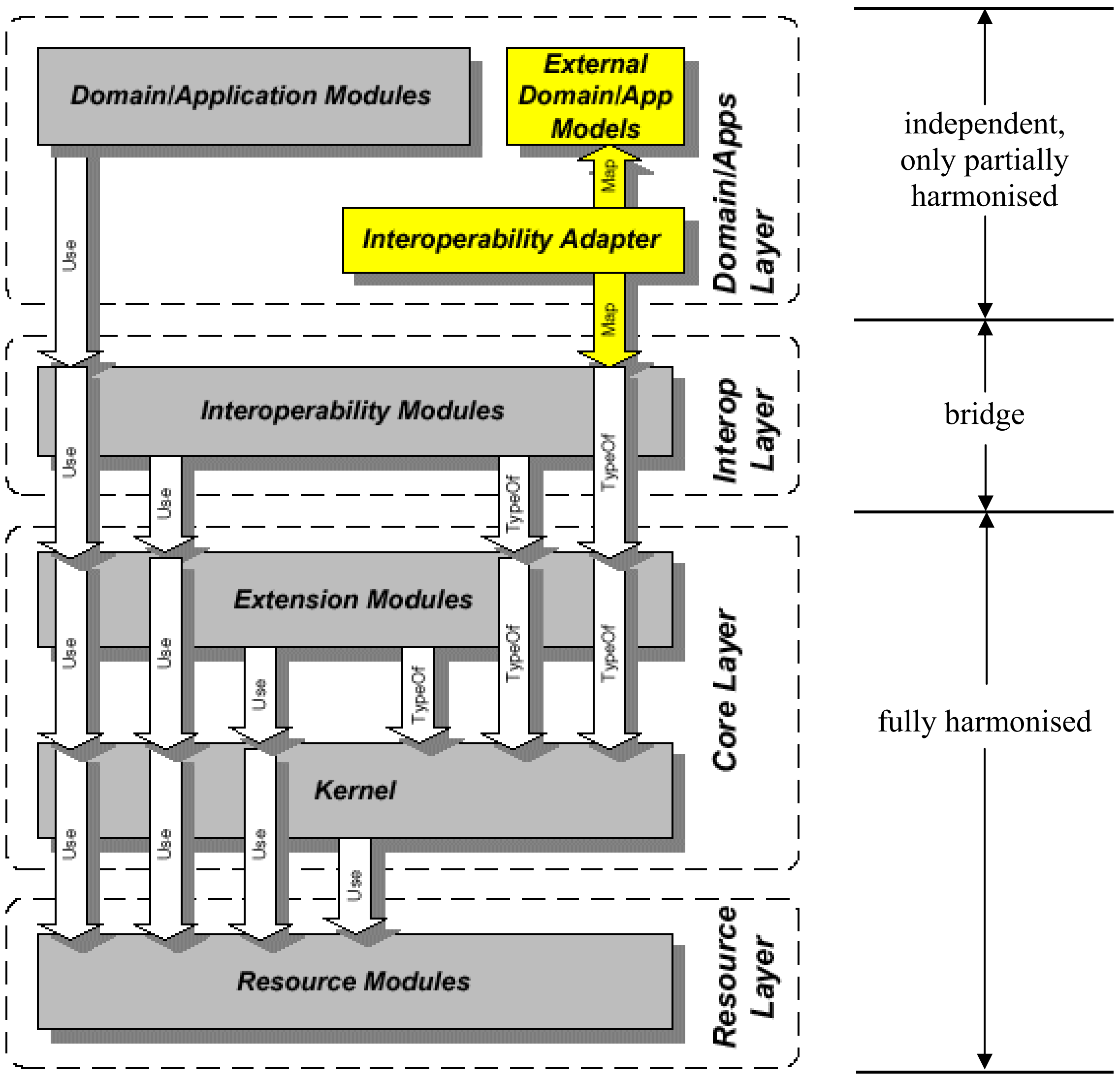


Fig. 5.2: The IFC layered model architecture / adapted from (IAI 1999b) /

As a whole, the IFC framework provides many useful concepts that can facilitate the semantic interoperability in an integrated concurrent engineering environment. It does not guarantee automatically that the domain data models will be fully harmonised, but it contributes to reduce as much as possible the occurrence of conflicts. However, even with

IFC as basis, many "technical" semantic interoperability problems remain open at the domain model level. These are:

- the very development of domain models[*)];
- the "horizontal" exchange of information between different discipline models which may contain non harmonised overlapping concepts;
- the possibly different semantic richness and the different definition of conceptually similar objects ("wall" is not like "wall" in architectural, structural, HVAC models).

5.5 Model Integration

The model integration strategy comes from the area of database research. In the last years, with the growing use of multiple databases, a number of different approaches for the definition of an integrated database schema for a set of partially harmonised (overlapping) schemas have been suggested. Though strongly varying in representational power and implementation details, all these approaches follow principally the same integration process (Conrad 1997):

1) Pre-integration phase (mainly manual)
   detection of all schema overlaps
2) Comparison of the overlapping data structures (mainly manual)
   detection of the inter-schema conflicts
3) Mapping (semi-automatic)
   definition and implementation of the inter-schema correspondences
4) Restructuring and merging (highly automated)
   rearranging, if necessary, of the information in the existing schemas
   and generation of the integrated model.

The emphasis of all model integration approaches is on the final (and partially the pre-final) of the above phases. The main objectives are to ensure high reliability and continuous consistency of the environment, understandability of the global model (which is supposed to be used directly by some applications) and minimal redundancy of the data.

These heavy requirements have strongly influenced the capacity of all proposed methods with the following consequences:

- most of the efforts have been directed towards a methodology for the design phase of the environment involving many manual operations, whereas in the run-time system data access, distribution and consistency are supposed to be managed automatically;
- the targeted application domains include mainly the integration of existing business or banking databases with relatively simple relational structures;
- some complex representational issues are intentionally sacrificed for the sake of better reliability and performance.

The selected approaches outlined below give an impression of the wide variety of proposed methods, but show also their common features as well as their drawbacks w.r.t. the overall solution strategy sought for CEE.

*) There is not much experience in the development of domain models at this time, but efforts like ST-2 (IAI ST-2 1998) show that even the harmonisation of models intended as "internal" to the IFC framework is not easy to achieve.

5.5.1 Superviews

Superviews (Motro 1987) is one of the earliest approaches for the virtual integration of multiple databases. The proposed method is based on a set of ten operations defining the transformations between the given source schemas and the "to be" *superview* schema. The goal is to generate and populate the "superview" model enabling bi-directional information transfer with the source models. The ten suggested operations are:

- *Meet*, producing a sub/superclass taxonomy from two overlapping entities having a common key;
- *Fold*, allowing to combine sub/super classes into a single generalised entity;
- *Aggregate*, creating a 1:N aggregation for an existing base entity and one or more of its complex (aggregate) attributes;
- *Telescope*, reducing the target structure by assigning the attributes of a referenced entity directly to the entity that references it;
- *Join*, producing a union of overlapping entities;
- *Combine*, allowing to merge two entities that have identical types into a new entity with a different name;
- *Connect* – as combine, but adopts the name of one of the original entities;
- *Add*, allowing the addition of new attributes to an entity, using a pre-defined function to assert their values;
- *Delete*, allowing to remove entities that are not relevant to the superview, and
- *Rename*, allowing to rename an entity without changing its content in any other way.

Superviews tackles all problems that can be described by these operations with guaranteed consistency, but offers no help for problems that fall outside their capacity. There are some provisions for handling sub/super relationships, but they require the availability of explicit common keys which are typical only for relational DBMS. Attributes derived by functional transformations, extracted from aggregate values or requiring 1:N or M:N correspondence between the source models and the superview are not supported at all. As such cases are typical for the complex modelling domain of CAE/CIC, Superviews cannot be used to solve the semantic interoperability problems in the targeted CEE, but it provides a structured method for handling certain mapping patterns which is worth considering as part of a more general model mapping approach.

5.5.2 Assertion based integration

Different variations of assertion based integration have been suggested in several research studies. The core of the original method, as described in (Spaccapietra et al. 1992) and (Spaccapietra & Parent 1994), is in the definition of *correspondence assertions* expressing the relationships between the data objects of the "to be" integrated model and the existing source models. Its main objective is to ensure the overall consistency in a heterogeneous multidatabase system containing data defined according to different representation paradigms. However, similar to Superviews, the assertion based integration approach can tackle mainly different variations of the relational paradigm; object-oriented models are more or less out of scope.

The method involves the following steps:

1) Converting each schema into a *canonical form* enabling the unified representation of three kinds of concepts: objects, value attributes and reference attributes;
2) Definition of the *correspondence assertions* for the elements of the base schemas, thereby distinguishing between object assertions, simple attribute assertions, complex attribute assertions and path correspondence assertions with the respective subtypes equivalence, inclusion, intersection and disjunction;
3) Identification and execution of *integration rules* that accept as input entities of two source schemas to produce an entity in the integrated schema; e.g. for the case $E1 \equiv E2$ the rule

   $E = \textit{integrate-join}(E1, E2, \mathit{attcor}_2(a_{11}, a_{22}), \ldots, \mathit{attcor}_j(a_{1j}, a_{2j}), \ldots))$

   creates a new object type E in the integrated schema using the bijective transformations defined by $\mathit{attcor}_j(a_{1j}, a_{2j})$ which guarantee the consistency of all instances of E, E1 and E2.

Best formalised and automated is the last of the above steps, whereas the greatest part of the first two steps has to be performed manually by the developer of the integrated model.

The assertion based integration approach is applicable to relational and Entity-Relationship models that do not contain M:N relations on attribute level. With some modifications it can be applied also to object-oriented representations that do not require the use of inheritance and bi-directional relationships as these are not supported. It is also not sufficient for the solution of the semantic interoperability problems in CEE, but provides several useful hints about certain types of mappings that need to be considered.

5.5.3 Formalised object-oriented integration

The object-oriented integration approach proposed in (Reddy et al. 1994) comprises a set of sophisticated formal procedures enabling the derivation of an integrated schema with guaranteed consistency and non redundancy of the data. Its primary objective is to provide a *lean global model* supporting queries for all "globally relevant" data, but hiding the details of the local models from the "global users" of the system. For that purpose, a 4-layer architecture is suggested as follows:

- *local schema layer*, containing the original source schemas intended for use by local applications;
- *local object schema layer*, containing model schemas that are semantically equivalent to the source schemas, but are represented according to the proposed object paradigm for the global model;
- *global object schema layer*, containing the integrated schema, and
- *global view layer*, containing external view schemas derived from the global model and intended for use by global applications.

The integration method consists of the following two principal steps:

1) Formal representation of all source schemas in a unified format defined by

   $\mathbf{S_i} = \{ \mathbf{O_i}, \Phi_i, \mathbf{SK_i}, \mathbf{MK_i} \}$

   where $\mathbf{S_i}$ is the respective source schema, $\mathbf{O_i}$ is the set of all its object types together with their attributes, Φ_i is a relationship matrix comprised of tupples $\phi_i = <\sigma, \delta>$ specifying four different kinds of relationships between the objects in $\mathbf{S_i}$ (subclass=1 and non subclass=0 for σ, and disjoint=1, overlapping=0 for δ), $\mathbf{SK_i}$ is the *semantic knowledge* about the schema describing, through meta attributes, the integrity conditions for all actual attributes, and $\mathbf{MK_i}$ is the *mapping knowledge* comprising the rules needed to convert each object $\mathbf{o_i} \in \mathbf{O_i}$ to the respective object $\mathbf{o_G} \in \mathbf{O_G}$ in the global schema.

2) Generation of the global schema $\mathbf{S_G} = \bigcup \mathbf{S_i}$ by using the above formalised representations of $\mathbf{S_i}$, along with a set of formal algebraic rules for the construction of the global relationship matrix $\Phi_G = \Sigma\ \Phi_i$, and interactive user input for those elements of Φ_G which represent structural conflicts that cannot be tackled automatically by the method.

As a whole, the approach is well suited for a subclass of object-oriented models that can be described by the developed formalism. It is also one of the few approaches that can provide almost complete non redundancy of the data in the integrated system. However, it cannot process several modelling concepts that are typical for the sophisticated product model world of CAE/CIC, including such important issues as multiple object references and object aggregations. As a consequence, this approach is also not suitable as an enabling method for CEE. Unfortunately, it is also of limited use for the development of formalised model mapping methods because the mapping knowledge component (MK) is not yet sufficiently developed.

In summary, all model integration approaches show that schema conflicts can be adequately tackled only by relatively simple data models. The biggest problems provide object/attribute aggregations which are more or less outside the scope of practically any of the known methods, but are quite important in the modelling of engineering products. On the other hand, the available features of today's databases enable the construction of large integrated systems, even if application interfaces have to be hand-coded. There are several known approaches utilising a sophisticated engineering database as the glue of a comprehensive design environment comprised of several heterogeneous applications (Encarnação & Lockemann 1990; Assal & Eastman 1995). In (Sauter & Käfer 1995) an interesting methodology for the use of EXPRESS models as the basis for such federated systems is suggested. In many situations this may well be the most adequate solution. However, in a concurrent engineering design environment it enforces an overall consistency which may be counterproductive as it can lead to unnecessarily sequentialised processes only because in a long transaction, e.g. when the structural system of a whole building is designed, a whole lot of data will need to be locked for hours, or even for days.

5.6 Model Mapping

In this section, the model mapping approach proposed as basis for the solution of the semantic interoperability problems addressed in this study will be analysed in detail. With this approach many of the difficulties experienced by the other presented approaches can be overcome, and a broader coverage of inter-schema conflicts can be achieved. This is due to the fact that by applying a model mapping strategy neither continuous consistency, nor non redundancy of the data need to be enforced to the overall system, which leads to "softer" integration requirements and clearly separated procedures for mapping, matching, consistency and integrity management. Thus, by model mapping, schema development must not be closely tied to prescribed harmonisation rules, and an integrated global model is not a necessary requisite of the environment, even though it would surely facilitate most of the mapping tasks.

As a whole, the overall procedure is very similar to the model integration process:

1) Detection of the schema overlaps	(mainly manual)
2) Detection of inter-schema conflicts	(mainly manual)
3) Definition of the inter-schema correspondences with the help of a formal mapping language	(highly formalised)
4) Use of appropriate mapping methods to perform the actual transformations on entity instance level when needed at run-time.	(highly automated)

By comparing the two procedures it can easily be recognised that there is a principal difference only in the 4th step. However, exactly this difference allows to develop *new*, more comprehensive *description methods* for the most essential part of the process, i.e. step 3.

At first, it must be noticed that the types of conflicts outlined in section 5.2 do not appear separate from each other. Rather, the mapping of a conceptual model, or even of one class in a model, is often related to several different conflict types which have to be solved coherently. Therefore, for the development of a mapping formalism it is necessary to analyse all components involved in a model transformation in their inter-relationship, i.e.:

– the *representation paradigms* used to describe the data models,
– the basic *set relational operations*,
– the principal *inter-schema mapping types*,
– *mapping patterns*, deduced from theoretical background knowledge and from the examination of typical practical cases, and
– basic *mapping language requirements*.

Here we shall assume that we have to deal primarily with EXPRESS-based data models. Thus, we shall not consider problems related to heterogeneity conflicts. Further, we shall assume that the fundamental relational algebra operations known from RDBMS have to be supported in the mapping of conceptual data models as well, because in many cases, especially when a model pre-harmonisation policy is pursued, the local models can be roughly seen as views of a more broader "integrated" data model[*)].

5.6.1 Formal representation of the basic EXPRESS data structures

A formal representation of the relevant data structures in an EXPRESS data model conforming to (ISO 10303-11 1994) is both a prerequisite for the development, and a key for the understanding of the basic mapping types and patterns discussed further in this section.

EXPRESS data models can be represented formally as follows:

Let $\mathbf{N}$ be the domain of all natural numbers, $\mathbf{N}$ be the set of all names in an EXPRESS schema ($\mathbf{S}$), $\mathbf{A}$ be the set of all correct attribute specifications, $\mathbf{E}$ be the set of all correct entity specifications, and $\mathbf{T(S)}$ be the set of all data types in a model. Then:

1. Each attribute specification in an entity class can be represented by the 4-tupple:

```
attr = < aname, atype, opt, d_expr >
```

where: $\mathbf{aname} \in \mathbf{N}$ is the attribute name
$\mathbf{atype} \in \mathbf{T(S)}$ is the attribute type
opt is a predicate which is `true` when the attribute is optional, and `false` otherwise
d_expr is an expression for the evaluation of derived attributes, with
$\mathbf{d\_expr} = \varnothing$ when the attribute is not derived, and
$\mathbf{d\_expr} = \text{exp}$, with $\text{type(exp)} = \mathbf{atype} \in \mathbf{T_{base}}$ when **attr** is derived.

In addition, $\forall\ \mathbf{attr} \in \mathbf{A}$ holds: $\mathbf{d\_expr} \neq \varnothing \Rightarrow \neg\mathbf{opt} \wedge \mathbf{atype} \in \mathbf{T_{base}}$.

*) This approach is well known from many research and development projects whose main objective has been the integration of design tools on the basis of a common integrated data model (IDM). Work in this direction has been conducted e.g. in the COMBINE project (Augenbroe 1995a), in the ATLAS project (Böhms & Storer 1994), by (Amor & Hosking 1995) etc. Although not explicitly stated, and not necessarily limited to that, the current IFC architecture assumes a similar approach.

2. An EXPRESS entity class can be defined formally by the 5-tupple:

ent = < ename, e_sup, e_attr, e_unique, e_lc >

where: **ename** $\in$ **N** is the name of the entity class
e_sup is the set of all direct superclasses, such that
e_sup = {e_sup$_1$, . . . ,e_sup$_n$ }, **e_sup** $\subset$ **E**, **ent** $\not\in$ superclass(**ent**)
e_attr is the set of all attributes of **ent**, such that
e_attr = {attr$_1$, . . . ,attr$_p$ }, and
$\forall$ i attr$_i$ $\in$ **A** , i $\in$ [1..p],
$\forall$ attr$_i$, attr$_j$ $\in$ attributes(ent), i $\neq$ j, i,j $\in$ [1..p] $\Rightarrow$
aname(attr$_i$) $\neq$ aname(attr$_j$)
e_unique is the set of all unique (non ambiguous) entity attributes, such that
e_unique = {e_unique$_1$, . . . , e_unique$_q$ }, and
$\forall$ i $\in$ [1..q] e_unique$_i$ $\in$ e_attr, q $\leq$ p
e_lc is the set of all local constraints on the values of the entity attributes, with
e_lc = {e_lc$_1$, . . . ,e_lc$_r$ } and $\forall$ i $\in$ [1..r] type(e_lc$_i$) = LOGICAL.

3. An EXPRESS schema **S** can be defined by the tupple:

S = < ent, R >

where: **ent** is the set of all entity specifications in **S**, such that:
ent = { ent$_1$, . . . ,ent$_n$ }, and
$\forall$ i $\in$ [1..n] ent$_i$ $\in$ **E**
$\forall$ ent$_i$,ent$_j$ $\in$ entity(**S**), i$\neq$j, i,j $\in$ [1..n] $\Rightarrow$ ename(ent$_i$)$\neq$ename(ent$_j$)
and **R** is the set of all global rules in the schema[*)].

4. The set of all data types **T(S)** in an EXPRESS schema **S** can be defined by:

T(S) = { T$_{base}$ $\cup$ T$_{aggr}$(S) $\cup$ T$_{constr}$(S) $\cup$ T$_{named}$(S) }

with:

T$_{base}$	=	{ **BOOLEAN, LOGICAL, INTEGER, REAL, STRING** }
T$_{aggr}$(S)	=	{ **T$_{array}$(S)** $\cup$ **T$_{set}$(S)** $\cup$ **T$_{bag}$(S)** $\cup$ **T$_{list}$(S)** }
T$_{array}$(S)	=	{t$_a$ | t$_a$ = <low,high,t>, low, high $\in$ **N**, low$\leq$ high, t $\in$ **T(S)**\t$_a$ }, where: 'low' and 'high' are the lower and upper boundaries of the array, and 't' is the type of the array elements
T$_{set}$(S)	=	{t$_s$ | t$_s$ = <min,max,t>, min, max $\in$ **N**, 0$\leq$ min$\leq$ max, t $\in$ **T(S)**\t$_s$ }
T$_{bag}$(S)	=	{t$_b$ | t$_b$ = <min,max,t>, min, max $\in$ **N**, 0$\leq$ min$\leq$ max, t $\in$ **T(S)**\t$_b$ }
T$_{list}$(S)	=	{t$_l$ | t$_l$ = <min,max,t>, min, max $\in$ **N**, 0$\leq$ min$\leq$ max, t $\in$ **T(S)**\t$_l$ } where: 'min' and 'max' are the minimum and maximum number of elements in the set, bag or list respectively, and 't' is the type of each element
T$_{constr}$(S)	=	{ **T$_{enum}$(S)** $\cup$ **T$_{select}$(S)** }
T$_{enum}$(S)	=	< enum_name,enum > where: enum_name $\in$ **N** is the name of the enumeration, enum = { t$_{en}$ | t$_{en}$ = < e$_1$, . . . ,e$_n$ > }, e$_i$ $\in$ **N**, i $\in$ **N**, and e$_i$ is the name of the ith enumeration type

*) Rules are not related to the data representation itself, but to the consistency of the data model. Therefore, they are not relevant to the mapping problems and are not considered further here.

$\mathbf{T_{select}(S)} = <\mathtt{sel\_name}, \mathtt{sel}>$
where: $\mathtt{sel\_name} \in \mathbf{N}$ is the name of the select type,
$\mathtt{sel} = \{ t_{sel} \mid t_{sel} = < s_1, \ldots, s_n > \}$,
$s_i \in \mathbf{T_{named}(S)}$, $i \in \mathbf{N}$

$\mathbf{T_{named}(S)} = \{ \mathbf{T_{def}(S)} \cup \mathbf{T_{ent}(S)} \}$

$\mathbf{T_{def}(S)} = <\mathtt{def\_name}, t_d>$
where: $\mathtt{def\_name} \in \mathbf{N}$ is the name of the defined type, and
$t_d \in \mathbf{T(S)}$, $t_d \notin \mathbf{T_{ent}(S)}$

$\mathbf{T_{ent}(S)} = \{t_e \mid t_e = \text{ename}(\mathtt{ent})$,
$\mathtt{ent} = \mathbf{<}\mathtt{ename, e\_sup, e\_attr, e\_unique, e\_lc}\mathbf{>} \in \mathbf{S} \}$.

5. The domain $\mathbf{D_{dt}}$ of a data type $\mathbf{dt} \in \mathbf{T(S)}$ depends on that type as follows:

if type($\mathbf{dt}$) = $\mathbf{BOOLEAN}$ $\Rightarrow$ $\mathbf{D_{dt}} = \{$ FALSE, TRUE $\}$

if type($\mathbf{dt}$) = $\mathbf{LOGICAL}$ $\Rightarrow$ $\mathbf{D_{dt}} = \{$ FALSE, TRUE, UNKNOWN $\}$

if type($\mathbf{dt}$) = $\mathbf{INTEGER}$ $\Rightarrow$ $\mathbf{D_{dt}}$ is the domain of all integer numbers

if type($\mathbf{dt}$) = $\mathbf{REAL}$ $\Rightarrow$ $\mathbf{D_{dt}}$ is the domain of all floating point numbers

if type($\mathbf{dt}$) = $\mathbf{STRING}$ $\Rightarrow$ $\mathbf{D_{dt}}$ is the domain of all valid strings

if type($\mathbf{dt}$) $\in \mathbf{T_{array}(S)}$ $\Rightarrow$ $\mathbf{D_{dt}} = d_1(t) \times \ldots \times d_n(t)$
with $d_i(t) = \mathbf{D_t}$, $i = 1..n$, $n = \text{high} - \text{low} + 1$ *)

if type($\mathbf{dt}$) $\in \mathbf{T_{set}(S)}$ $\Rightarrow$ $\mathbf{D_{dt}} = \{ s \mid s = \{t_1 \ldots t_n\} \}$
with $t_i \in \mathbf{D_t}$, $i = 1..n$, $\min \leq n \leq \max$ *)

if type($\mathbf{dt}$) $\in \mathbf{T_{bag}(S)}$ $\Rightarrow$ $\mathbf{D_{dt}} = \{ b \mid b = \{b_1 \ldots b_n\} \}$
with $b_i = < \eta_i, t_i >$, $\eta_i \in \mathbf{N}$, $t_i \in \mathbf{D_t}$, $i = 1..n$,
$\min \leq \eta_1 + \ldots + \eta_n \leq \max$ *)

if type($\mathbf{dt}$) $\in \mathbf{T_{list}(S)}$ $\Rightarrow$ $\mathbf{D_{dt}} = \{ lst \mid lst = \{e_1 \ldots e_n\} \}$
with $e_i \in \mathbf{D_t}$, $i = 1..n$, $\min \leq n \leq \max$ *)

if type($\mathbf{dt}$) $\in \mathbf{T_{enum}(S)}$ $\Rightarrow$ $\mathbf{D_{dt}} = \mathtt{enum}$

if type($\mathbf{dt}$) $\in \mathbf{T_{select}(S)}$ $\Rightarrow$ $\mathbf{D_{dt}} = \mathtt{sel}$

if type($\mathbf{dt}$) $\in \mathbf{T_{def}(S)}$ $\Rightarrow$ $\mathbf{D_{dt}}$ is the domain of the underlying type t_d

if type($\mathbf{dt}$) $\in \mathbf{T_{ent}(S)}$ $\Rightarrow$ $\mathbf{D_{dt}} = \{ e \mid e \in \mathbf{D_{ent}} \}$, $\mathtt{ent} \in \mathbf{E}$.

6. The domain $\mathbf{D_{ent}}$ of an entity class $\mathbf{ent} \in \mathbf{E}$, specified by
$\mathtt{ent} = \mathbf{<}\mathtt{ename, e\_sup, e\_attr, e\_unique, e\_lc}\mathbf{>}$ and
attributes($\mathtt{ent}$) $= \{ \mathtt{attr}_1, \ldots, \mathtt{attr}_n \}$,
can be defined as:

$$\boxed{\mathbf{D_{ent}} = \mathbf{D}\,(\ \text{type}(\mathtt{attr}_1) \times \ldots \times \text{type}(\mathtt{attr}_n)\ )}$$

7. Finally, an instance $\mathbf{e_i}$ of an entity class $\mathbf{ent}$ can be defined as:
$\mathbf{e_i} = [\ a_1, \ldots, a_n\ ]$,
where: $\forall\ i \in [1..n]\ \ a_i \in \mathbf{D}_{\text{type}(\mathtt{attr}_i)}$.

*) see definition 4 above for low, high, min, max and t respectively.

5.6.2 Set relational operations

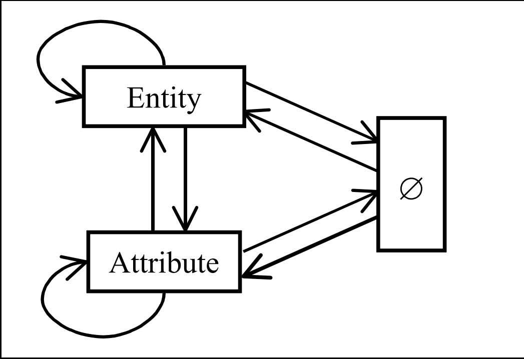
Pure relational data models have more limited representational power compared to object-oriented models. However, different variations of relational algebra operations can be encountered in many model transformation problems as well, especially when the target model has the typical characteristics of a database view w.r.t. the source model. Therefore, the representation of such operations must also be considered in the development of a model mapping formalism.

The basic set relational operations used in RDBMS are *projection*, *selection*, *cartesian product*, *union* and *set difference*. With their help a number of commonly used higher order operations, such as *intersection*, *join*, *natural join* etc. can be derived. All these operations are covered in detail in the database literature (Ullman 1988; Codd 1990) and will not be discussed further here. Later on, in chapter 6, it will be shown how these operations are supported by the mapping formalism developed in this thesis.

5.6.3 Principal mapping types

Although mapping problems have been intensively examined over the last ten years, there does not yet exist a solid understanding, neither a common formalism to define the basic mapping types involved in model transformation and model integration tasks.

Bijnen (1995) draws eight basic mapping types that might occur in a typical model transformation problem, all of which fall in the category "structural conflicts" (see fig. 5.3).



Entity → Entity	
Attribute → Attribute	
Entity → Attribute	(entity destructuring)
Attribute → Entity	(entity construction)
Entity → ∅	(entity deletion)
Attribute → ∅	(attribute removal)
∅ → Entity	(entity creation)
∅ → Attribute	(attribute addition)

Fig. 5.3: Basic mapping types / after (Bijnen 1995) /

The above figure provides a good overview of the types of structural conflicts that have to be dealt with in principle, but it does not give many hints as to what language formalisms and mapping techniques are needed to handle each of the presented cases. More detailed analyses of structural mapping types, taking in consideration the involved data types and the cardinalities of the inter-schema relationships have been made in (van Horssen et al. 1994; Katranuschkov 1995; Amor 1997). On the basis of these analyses, table 5.1 showing the full set of structural mapping types has been derived. In this table all meaningful combinations for entity classes, entity instances and entity attributes with cardinalities of zero (**0**), one (**1**), constant greater than 1 (**C**) and variable numbers (**N**) have been taken into account. To show that many-to-many mappings can involve different cardinalities for the source and the target, the letters **B** and **M** are used instead of **C** and **N** in two cases.

Table 5.1: Full set of structural mapping types

Cardi-nality	Class → Class	Entity Inst. → Entity Inst.	Attr. → Attr.	Entity Inst. → Attr.	Attr. → Entity Inst.
0 : 1	+	+	+		
1 : 0	+	+	+		
1 : 1	+	+	+	+	+
1 : C	+	+	+	+	+
1 : N		+			+
C : 1	+	+	+	+	+
C : B	(+)	(+)	+	+	+
C : N		(+)			+
N : 1		+		+	
N : C		(+)		+	
N : M		(+)			

The main differences between this table and the analysis presented in (Amor 1997) are in the added column representing the mapping types on class level, as well as in the mapping types given in parenthesis (+).

The mappings on class level are actually a subcategory of the mappings on instance level, but they are also the starting point for the definition of a mapping formalism and should therefore be considered separately. Besides, in pre-harmonised models it would not be unusual to specify many of the needed model transformations entirely on that level, without additional filtering or selection conditions for entity instances, and thus it is desirable to have a short and expressive format for such cases.

The parenthesized mapping types in the table can be avoided if both the source and the target model are constructed so that each N:M relationship is broken down to N:1 and 1:M relationships with the help of *relationship objects*. This approach is in fact consistently supported in the IFC Project Model by a subtree of entity classes with the abstract superclass *IfcRelationship* as their root. Probably because of a similar pre-sumption, these mapping types have not been considered necessary in the ATLAS project (van Horssen et al. 1994), and were not realised in practice in the COMBI project as well, even though a small example of a N:M mapping has been successfully tested (Katranuschkov & Scherer 1995).

A comprehensive checklist of inter-schema conflict types applicable to relational schemas has been given by (Kim & Seo 1991). This checklist, shown in table 5.2 below, has been used often as basis for the consideration of multidatabase integration issues in relational DBMS. However, even if the presented conflict types in this table appear quite detailed, there are almost no new mapping types to be found in it. For example, it is easy to see that the conflict types I.A.1.a and I.B.1.a are only more detailed descriptions of certain commonly met descriptive conflicts, the conflict types I.A.1.b, I.B.1.b and I.B.1.c are in fact specific structural conflicts etc. The conflict type II.A ("wrong data") is a new category, quite important for the integration of existing database systems, but with little relevance to model transformation problems because (1) many such incorrect data cannot be detected anyway as they do not necessarily lead to formal inconsistencies in the models, and (2) the consideration of such kinds of conflicts is not really in the scope of a mapping task, but should eventually be tackled by a consistency checking module, prior to a mapping. Thus, the only interesting new conflict types to be considered are the ones listed in II.B. Item II.B.1 addresses the problems associated with *type/value conversions*, such as `integer` → `real`, `0` → `"insufficient"`, `1` → `"sufficient"` etc. Item II.B.2 covers the problems associated with *unit conversions*, such as $kN/m^3 \rightarrow kg/cm^3$, and item II.B.3 covers problems associated with different *numeric precisions*, which is more an implementational, than a conceptual issue.

As a whole, table 5.2 can be quite helpful for estimating the difficulty of a particular mapping problem, but it is also not sufficiently detailed to provide the necessary hints for the development of a mapping formalism. For that purpose, a more detailed micro-level analysis of common *mapping patterns* which can be used as “building blocks” in the development of a mapping specification needs to be performed.

Table 5.2: Conflict types between relational model schemas /after (Kim & Seo 1991) /

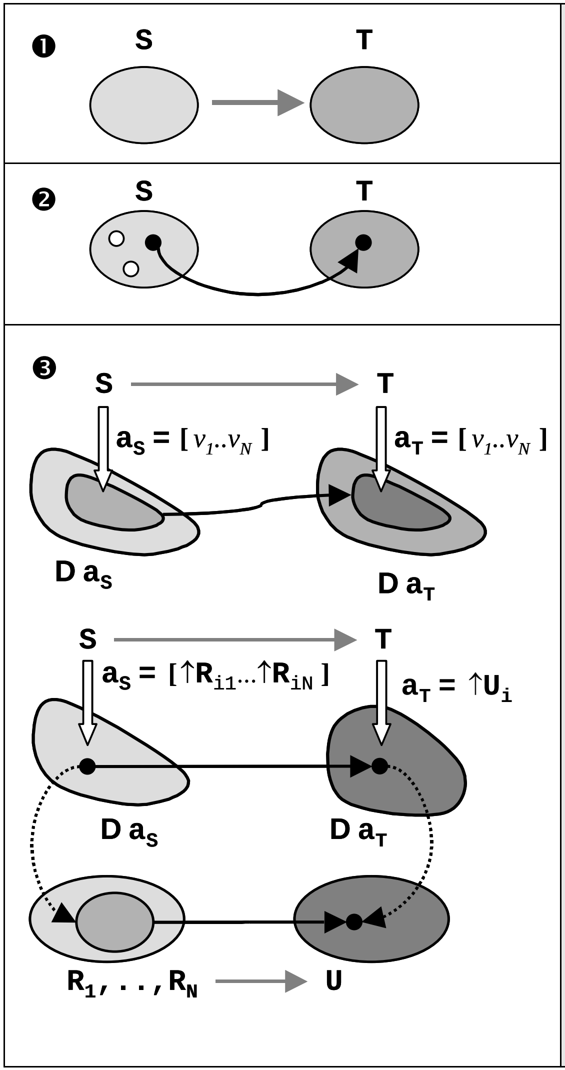
I. Schema conflicts		
	A. Table vs. table conflicts	
		1. One-to-one table conflicts a. Table name conflicts different names for equivalent tables same name for different tables b. Table structure conflicts missing attributes missing but implicit attributes c. Integrity constraint conflicts 2. Many-to-many table conflicts (as in one-to-one)
	B. Attribute vs. attribute conflicts	
		1. One-to-one attribute conflicts a. Attribute name conflicts different names for equivalent attributes same name for different attributes b. Default value conflicts c. Attribute constraint conflicts data type conflicts attribute integrity constraint conflicts 2. Many-to-many attribute conflicts (as in one-to-one)
	C. Table vs. attribute conflicts	
II. Data conflicts		
	A. Wrong data	
		1. Incorrect entries 2. Obsolete data
	B. Different representations	
		1. Different expressions 2. Different units 3. Different precision

5.6.4 Novel mapping patterns

The identification of mapping patterns has not been addressed in any known analyses of mapping problems. However, typical mapping patterns can provide a lot of clues for the development of a mapping formalism because they allow for better understanding of what and how has to be mapped in each particular case.

The patterns presented on the next pages (table 5.3 to table 5.7) have been developed by the author by taking into account the theoretical considerations from the preceding sections, as well as several examined practical examples. The first two tables, depicting in some more detail the mapping types “Class → Class” and “Entity instance → Entity instance”, are shown mainly for completeness. The other three tables are dedicated to the examination of different *attribute level* mapping patterns. They are related to the modelling concepts of EXPRESS data models, but can be used with most of the concepts of other object-oriented modelling paradigms as well. They reveal a new perspective to the development of attribute mappings which are at the heart of each practical mapping task.

For better readability, all patterns are shown schematically, with the help of the following, newly introduced graphical symbols (*MAPPING-G*):



❶ ***Class level mappings***
S and **T** (eventually with a subscript) are used to denote the source and the target class respectively. The ovals show the domains of the classes, the grey arrow shows the direction of the mapping. The given example depicts a 1:1 mapping **S** → **T**.

❷ ***Conditional entity instance level mappings***
In addition to the above symbols, here the black dots represent the instances being mapped, and the white dots represent the instances that are discarded from the mapping as a result of an applied *condition*. The particular mapping equivalences are shown with black arrows.

❸ ***Attribute level mapping patterns***
For these patterns, a few more graphical symbols are introduced. To save space, the class mapping (**S**→**T**) is indicated in most cases merely with a grey arrow. The domains of the attributes (**Da**) are shown as free-form areas, and the attribute values are shown either with dots (for single values), or with a darker area contained in the domain (for lists or sets). As above, the addressed mapping equivalences are indicated with black arrows. The thick white arrows denote the attribute’s designation to a given class (**S**, **T**); *pointer references* to other objects within a model are shown with dotted arrows. The first given example depicts a simple 1:1 attribute mapping, where both attributes are of aggregate data type. The second example shows a complex equivalence, involving two pairs of mappings due to the references from **a_S** and **a_T** to the object instances **[R_1, . . , R_N]** and **U** respectively.

Fig. 5.4: Graphical symbols used for the presentation of mapping patterns

Table 5.3: Unconditional class level mapping patterns

Note: The patterns in this table depict the most general high-level mappings that affect *all* instances of the involved source and target classes.

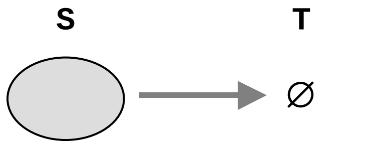
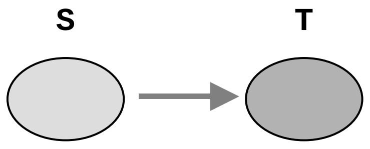
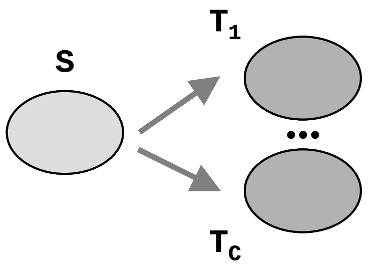
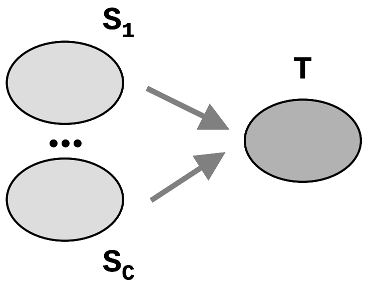
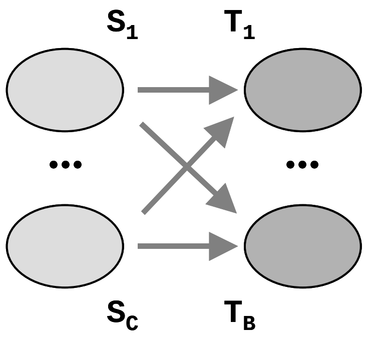
Class mapping pattern	Schematic presentation of the mapping transformations	Description
1 : ∅		Class deletion, i.e. no transformation of any instances from the source to the target model. As an example for the use of this pattern in IFC consider an "architectural" subclass of *IfcRelationship* that might not be needed in HVAC or structural models and is therefore excluded from the mapping.
1 : 1		Deep copy of a class. All instances of the source class are copied to the target "as is", including all references to other objects. Example for IFC: Since the components of the IFC model architecture are strongly pre-harmonised, this mapping will in fact be applicable for many of the IFC classes. "Simple" resource objects like *IfcPoint* would probably always be mapped in this way.
1 : C		Splitting of one source class into two or more different classes in the target. This pattern may be needed often when a mapping from an application-specific model to an IFC domain model is required. For example, the properties of a building material are often represented in applications as a single object structure, whereas in IFC this involves two or more entities.
C : 1		Combining of two or more source classes into a single class in the target. This pattern presents the inverse case to "splitting". As an example, consider the combining of "beam" and "column" entities into a "frame" entity for the purposes of a structural domain model.
C : B		A multiple C:B mapping involving both splitting and combining of the source classes. The number of source model classes must not necessarily be the same as the number of the target model classes participating in the mapping.

Table 5.4: Conditional instance level mapping patterns

Note: In contrast to the general definition of mappings on class level, instance level mappings may include certain conditions to select the set of instances to be mapped from the full set of available instances in the source model, or – in the case of multiple cardinality – from the cross product of all affected sets of instances.

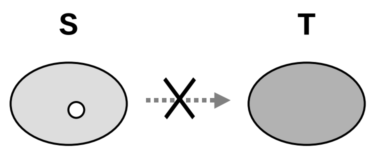
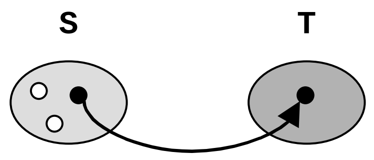
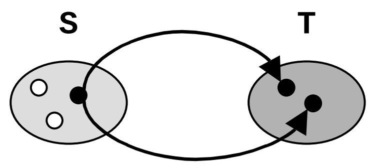
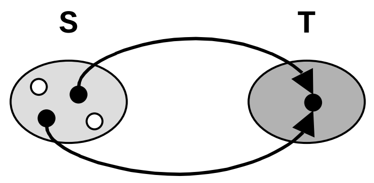
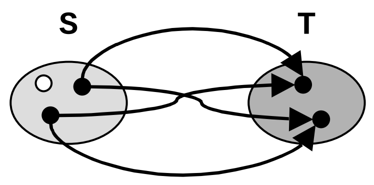
Instance mapping pattern	Schematic presentation of the mapping transformations	Description
1 : ∅		Entity deletion. Depending on one or more conditions, one or more source entity instances are *not* mapped to the target model. A typical example for the use of this pattern in IFC is the discarding of all walls that do not have the property 'bearing' in the mapping of an architectural to a structural domain model.
1 : 1		Deep copy of an entity. Depending on zero or more conditions, the affected entity instances are copied to the target model, including all value and reference attributes. If the dependent attribute mappings involve complex relations in the source and/or target model, the attribute level mapping patterns detailed in tables 5.5 to 5.7 have to be considered in addition.
1 : C 1 : N		Splitting of one source entity instance into two or more entity instances in the target. Unlike the similar in structure class mapping pattern from table 5.3, here the target instances may belong to the same or to different classes. The iteration process creating multiple instances in the target model will as a rule be governed by one or more of the involved attribute level mappings.
C : 1 N : 1		Grouping of two or more source entity ins-tances into one target instance. This is in effect the inverse operation to the above; hence, the same remark w.r.t. table 5.3 applies.
C : B C : N N : C N : M		Splitting and grouping. This operation com-bines the effect of the above two patterns. It can involve constant or varying number of instances, depending on the selection criteria and/or the involved detailed attribute mapping patterns.

Table 5.5: Basic attribute level mapping patterns

Note: Attribute level mapping patterns depict how an attribute with a given data type should be mapped. As shown in the given formal representation of EXPRESS, the data type of an attribute can be quite complex. Thus, there are many different patterns that have to be considered. In several cases they can also govern the actual resulting cardinality of the mapping on entity level. In this table, only the basic attribute patterns are shown; more sophisticated cases are treated in table 5.6 and in table 5.7.

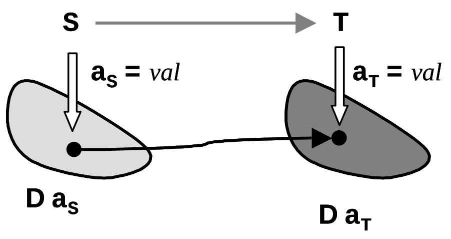
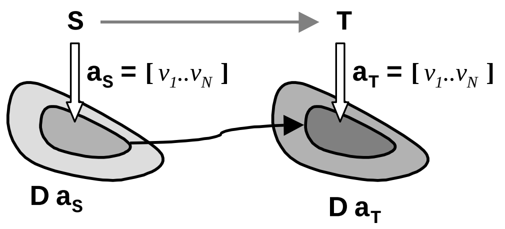
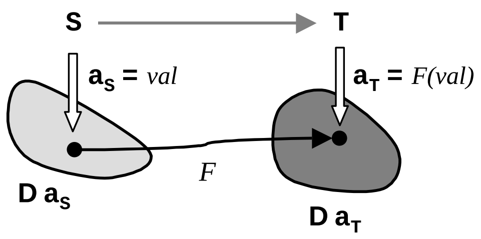
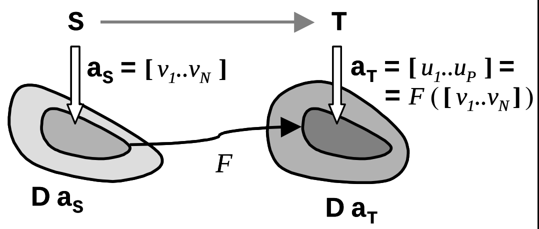
Attribute mapping pattern	Schematic presentation of the mapping transformations	Description
Identity (simple equivalence)	 S T a_S = *val* a_T = *val* D a_S D a_T	The simplest attribute mapping pattern is the 1:1 copying of a value attribute from the source to the target. In the strongly pre-harmonised IFC models, as well as in the case of deriving a view model from a more sophisticated source model, this simple mapping will be quite common to use.
Aggregate identity (simple set equivalence)	 S T $a_S = [v_1..v_N]$ $a_T = [v_1..v_N]$ D a_S D a_T	This is a similar case to the above, but for value attributes having an aggregate data type. Although the involved attributes may contain multiple values, this does not influence the cardinality of the entity mapping.
Functional equivalence	 S T a_S = *val* a_T = *F(val)* *F* D a_S D a_T	Here, the mapping of a source attribute to the target is not done by copying, but is the result of a *mapping function*. In principle, this function can be of arbitrary complexity, but quite often it will be just a simple expression, such as the computation of cross section properties (A, Ix, Iy etc.) from given dimensions and shape.
Functional set equivalence	 S T $a_S = [v_1..v_N]$ $a_T = [u_1..u_P] =$ $= F([v_1..v_N])$ *F* D a_S D a_T a) **N = P** b) **N ≠ P**	This pattern depicts the functional mapping of aggregate data types. Similar to the other basic patterns, it also does not affect the cardinality of the overall entity mapping. However, because a_S and a_T can be lists or sets, the function *F* may change the number of elements in the target. This is indicated by the two sub-cases a) and b).

Table 5.6: Complex attribute level mapping patterns

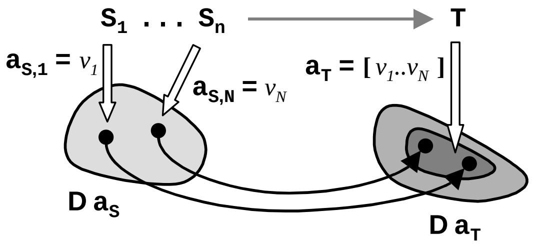
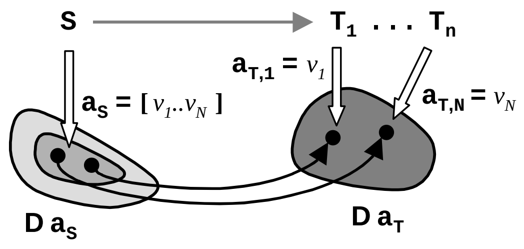
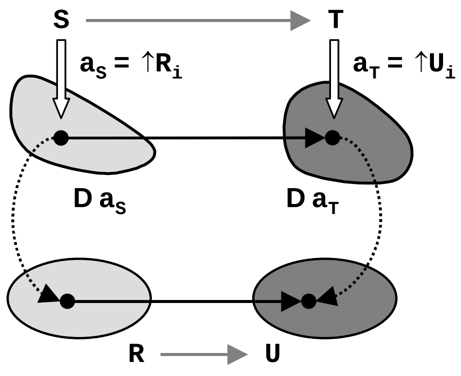
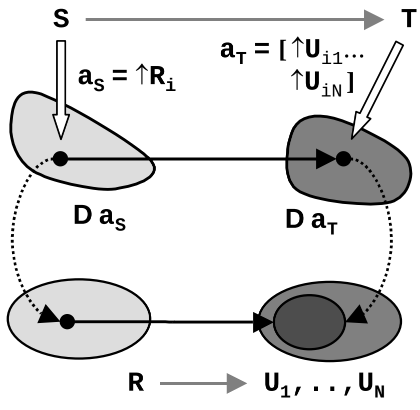
Attribute mapping pattern	Schematic presentation of the mapping transformations	Description
Grouping	 $S_1 \dots S_n$ T $a_{S,1} = v_1$ $a_{S,N} = v_N$ $a_T = [v_1..v_N]$ $D\ a_S$ $D\ a_T$	This pattern depicts the mapping of a source attribute from several source instances to one target instance. It defines implicitly a N:1 entity mapping. The involved source instances can be a subset of all available instances as a result of one or more applied conditions. The target attribute must be of an aggregate data type. As an example, consider the grouping of the materials of building elements to create a "bill of materials" object.
Ungrouping (iteration)	 S $T_1 \dots T_n$ $a_{T,1} = v_1$ $a_S = [v_1..v_N]$ $a_{T,N} = v_N$ $D\ a_S$ $D\ a_T$	This is the inverse case to the above: the mapping is of cardinality 1:N, the source attribute must be of an aggregate data type. A typical example of this pattern can be seen in the case study described in section 7.1.
Homo-morphic (1:1 assoc.)	 S T $a_S = \uparrow R_i$ $a_T = \uparrow U_i$ $D\ a_S$ $D\ a_T$ R U	This and the next three patterns represent "paired" homomorphic mappings. The involved attributes $\mathbf{a}_S$ and $\mathbf{a}_T$ are both reference attributes. Therefore, the mapping is dependent on the mapping of the referenced instances **R** and **U** respectively. The cardinality of the mapping $\mathbf{S} \rightarrow \mathbf{T}$ is not affected, the given case (1:1) refers to the cardinality of the mapping $\mathbf{R} \rightarrow \mathbf{U}$.
Homo-morphic (1:N assoc.)	 S T $a_T = [\uparrow U_{i1} \dots \uparrow U_{iN}]$ $a_S = \uparrow R_i$ $D\ a_S$ $D\ a_T$ R $U_1, \dots, U_N$	In this case, because of the 1:N mapping $\mathbf{R} \rightarrow \mathbf{U}$, a single valued source attribute is transformed to an attribute of aggregate data type in the target. As an example, consider the already described situation of single object "material" description in an application model vs. multiple objects used in IFC for that purpose. The mapping of any entity references to materials will be in this category.

Table 5.6 (cont.): Complex attribute level mapping patterns

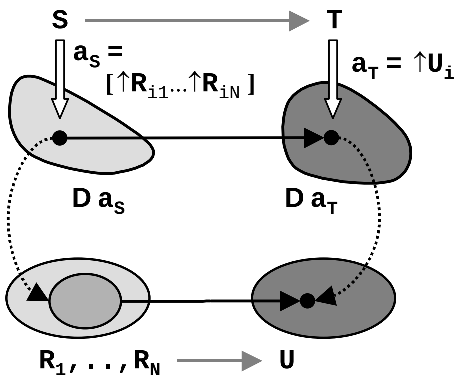
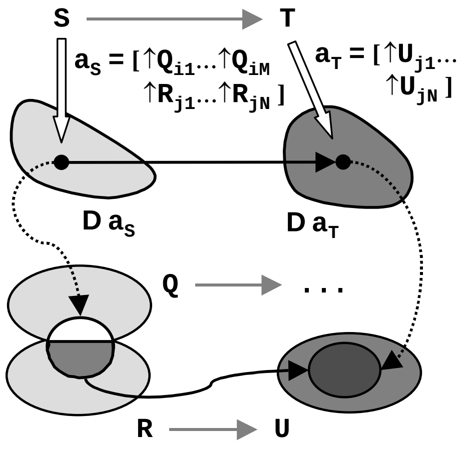
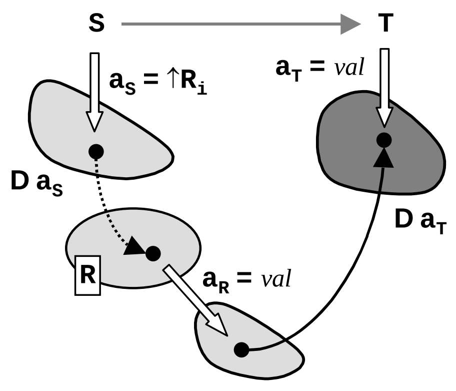
Attribute mapping pattern	Schematic presentation of the mapping transformations	Description
Homo-morphic (N:1 assoc.)	 S T $\mathbf{a_S} = [\uparrow\mathbf{R}_{i1} \ldots \uparrow\mathbf{R}_{iN}]$ $\mathbf{a_T} = \uparrow\mathbf{U}_i$ D $\mathbf{a_S}$ D $\mathbf{a_T}$ $\mathbf{R_1, \ldots, R_N}$ U	The third homomorphic pattern is the inverse to the second: an aggregate source attribute is "collapsed" to a single pointer value in the target. As an example, consider a situation where a *composite beam* is represented with the help of several "section" entities in the source, which are transformed to a single "property" object in the target.
Homo-morphic selective	 S T $\mathbf{a_S} = [\uparrow\mathbf{Q}_{i1} \ldots \uparrow\mathbf{Q}_{iM}\ \uparrow\mathbf{R}_{j1} \ldots \uparrow\mathbf{R}_{jN}]$ $\mathbf{a_T} = [\uparrow\mathbf{U}_{j1} \ldots \uparrow\mathbf{U}_{jN}]$ D $\mathbf{a_S}$ D $\mathbf{a_T}$ Q . . . R U	The last considered homomorphic pattern is typical for situations where a SELECT type referencing different classes is involved. There can be several variations of this pattern. In the given schematic presentation on the left, the source attribute **a_S** contains pointers to instances of two different classes (**Q** and **R**) of which only **R** must be considered in the mapping of **a_S** to **a_T**. Because in IFC such SELECT types are numerous, in the mapping to an application model this pattern may be needed quite often. As an example, consider a mapping of *IfcActorSelect*, which can include both persons and organizations, to a model where only persons are of interest.
Transitive (telescope)	 S T $\mathbf{a_S} = \uparrow\mathbf{R}_i$ $\mathbf{a_T} = val$ D $\mathbf{a_S}$ D $\mathbf{a_T}$ R $\mathbf{a_R} = val$	This pattern is well-known from the field of database integration. It represents a situation where the value of a source attribute **a_R** referenced through a pointer in **a_S** is stored directly into the target attribute **a_T**. The schema on the left presents the most common case of a single reference used to access **a_R**, but in general such references can also be chained.

Table 5.6 (cont.): Complex attribute level mapping patterns

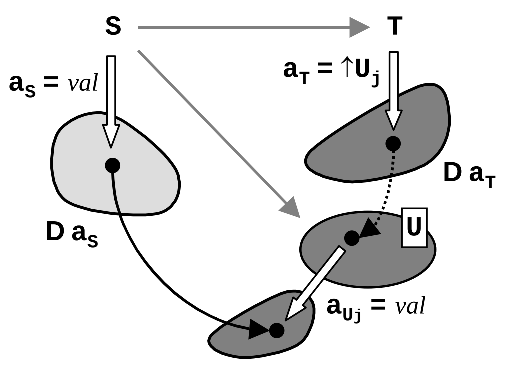
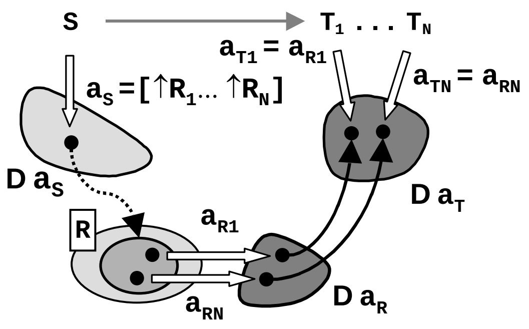
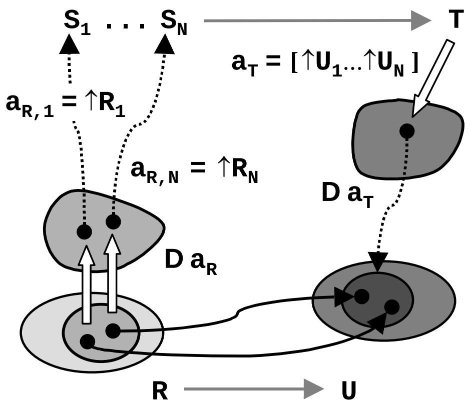
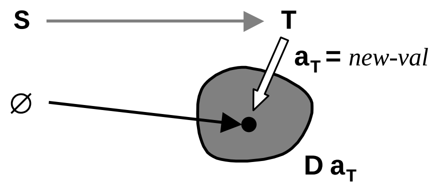
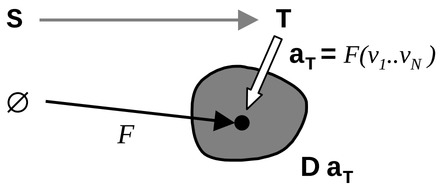
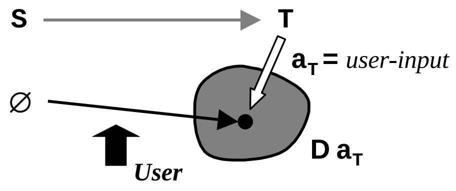
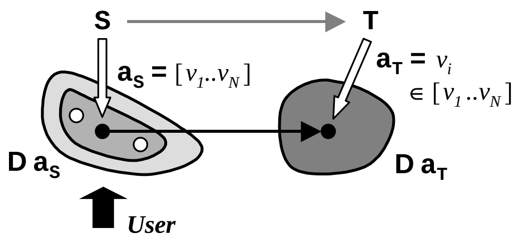
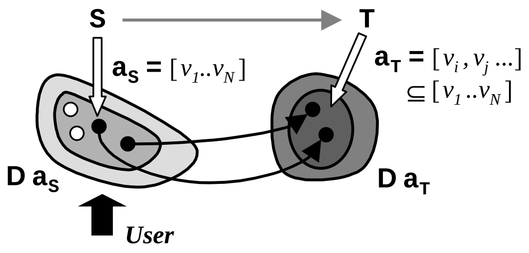
Attribute Mapping Pattern	Schematic presentation of the mapping transformations	Description
Inverse transitive	 S T a_S = *val* $a_T = \uparrow U_j$ D a_S D a_T U a_{Uj} = *val*	This pattern is the inverse of the previous one. For conciseness, inverse patterns are not shown for all meaningful cases, but the inverse transitive pattern deserves more attention. Its realisation is much more difficult than by the forward mapping because, in order to create the value $\mathbf{a_T} = \uparrow \mathbf{U_j}$, all referenced instances have to be constructed in the target *before* the value of $\mathbf{a_T}$ can be properly set.
Transitive and associative	 S $T_1 ... T_N$ $a_{T1} = a_{R1}$ $a_S = [\uparrow R_1 ... \uparrow R_N]$ $a_{TN} = a_{RN}$ D a_S D a_T R a_{R1} a_{RN} D a_R	As prompted by its name, this rather complex pattern is in fact a combination of two other patterns that have already been presented. It is included because of its relatively frequent occurrence in the author's experience gathered from examined practical mapping tasks.
Inverse associative	 $S_1 ... S_N$ T $a_T = [\uparrow U_1 ... \uparrow U_N]$ $a_{R,1} = \uparrow R_1$ $a_{R,N} = \uparrow R_N$ D a_T D a_R R U	The last identified complex pattern is also derived by experience. It depicts a complex situation in which the reference pointers in the source are in opposite direction w.r.t. the target. This is the case when a *part-of* relation is represented as *has-parts* in the target and no inverse relationships can be taken into account. A specific example of this pattern is shown in section 7.2, by the mapping of the pair of source entity classes: *building–foundation_element* in a foundation design model, to the respective pair of target classes *structural_system–structural_element* in a structural domain model.

Table 5.7: Generative attribute level mapping patterns

Note: All generative patterns are related to the mapping type "∅ → `Attribute`", i.e. they describe different alternatives for the assignment of values to attributes of instances in the target model that do not have corresponding counterparts in the source model. All such attributes have to be defined in advance in the target model schema. The addition of new attributes changing the class definitions and the creation of new classes are not considered in the scope of the mapping approach as this can lead to run-time consistency problems that are hardly possible to solve. Such tasks are part of the model development process, but not of the practical data management in a project.

Attribute mapping pattern	Schematic presentation of the mapping transformations	Description
Simple generative	 S T $\mathbf{a_T}$ = *new-val* ∅ D $\mathbf{a_T}$	Assignment of predefined constant value(s). This pattern is rather rare. An example could be the specification of normative modules of elasticity for predefined building materials.
Functional generative	 S T $\mathbf{a_T} = F(v_1..v_N)$ ∅ F D $\mathbf{a_T}$	Assignment of value(s) to the target attribute $\mathbf{a_T}$ by a function. The function may use as arguments user input and/or data from the source model.
User-dependant generative	 S T $\mathbf{a_T}$ = *user-input* ∅ User D $\mathbf{a_T}$	Assignment of a value to the target attribute $\mathbf{a_T}$ by the user. This pattern can be useful when only a few data provided by the user would complete a model transformation. E.g., consider the definition of an overall *foundation depth* by the mapping of structural results to provide input data for foundation design.
User-dependent selective	 S T $\mathbf{a_S} = [v_1..v_N]$ $\mathbf{a_T} = v_i \in [v_1..v_N]$ D $\mathbf{a_S}$ D $\mathbf{a_T}$ User	Assignment of a single value to the target attribute $\mathbf{a_T}$ which is selected by the user from a list of possible values, typically contained in an enumeration defined in the target model, or specified in the mapping model itself.
User-dependent multiple selective	 S T $\mathbf{a_S} = [v_1..v_N]$ $\mathbf{a_T} = [v_i, v_j ...] \subseteq [v_1..v_N]$ D $\mathbf{a_S}$ D $\mathbf{a_T}$ User	This pattern is a common variation of the above. It depicts the assignment of *multiple* values to the target attribute $\mathbf{a_T}$ selected by the user from a list of possible values.

5.6.5 Mapping language requirements

As shown in the previous section, most of the model transformation problems are associated with the various patterns of attribute relationships within and between the involved schemas. In principle, these transformations can be represented graphically, but their details, such as the specification of conditions, value conversions, various correspondence assertions etc., have to be expressed in *textual form*. Unfortunately, there exists no formal method enabling the strict definition of requirements for a specification language of that kind. Therefore, an informal set of requirements is derived from the presented analysis of the different relevant aspects of the problem, the study of previous work, and the examination of existing actual conceptual model schemas, as follows:

1) *High-level notation*
   The definition of a mapping is likely to involve a large amount of work, demanding considerable efforts of experts both in conceptual modelling and in one or more technical domains. Usually, these experts are not qualified programmers and should not be bothered about the technicalities of the mappings in terms of program code. The design of high-level notational constructs, reflecting closely the representation paradigm of the involved models, is therefore of primary importance.

2) *Declarative style*
   The rationale for specifying the requirement for declarative language style is based on the following considerations:
   - to provide domain experts with an instrument enabling them to *declare* the needed mapping transformations, without forcing them to devise outlines or flow charts for implementation, for which they might not possess the necessary expertise (this is similar to the rationale for rule-based vs. procedural design by expert systems);
   - to separate the specification of a mapping task from its realisation in order to enable the use of different programming languages and implementation styles for the mapping engine responsible for the actual execution of the mappings in the running system;
   - to isolate the use of functional transformations requiring procedural code from the more general and more frequently needed pure set relational operations;
   - to use as little as possible procedural code (such code tends to be "less generic" and would bind the language tighter to a particular programming paradigm).

3) *Completeness*
   The mapping language should be capable to reference all data types in a conceptual model and to support all mapping types and patterns within the scope identified on the previous pages of this section.
   However, as there exists no formal methodology to define and describe all possible relationships within and between object-oriented model schemas, a mapping language can be "complete" only in a relative sense. Hence, a more practical interpretation of this requirement would be to focus on providing comprehensive support to existing and emerging EXPRESS-based data models in the AEC domain, with emphasis on IFC-related modelling efforts.

4) *Modularity*
   Mappings may be needed both for full (all data) or partial (selected entities) model transformations, in different execution modes (batch, interactive), and using different communication mechanisms (remote procedure calls, file transfer, CGI-scripts etc.). The mapping language should support a modular implementation approach enabling the realisation of any of these alternatives.

5) *Understandability*
   The syntax of the language has to be understandable to model and system designers as well as to end-users not only for development, but also for reviewing and maintenance purposes. Therefore, it must use a terminology which is amenable to conceptual modelling specialists and engineers. Conciseness of expression, as in Java, C or C++, is a secondary issue because mapping specifications would never be as long as actual executable programs. In this respect, a similarity to EXPRESS and high-level symbolic languages like LISP can be envisaged.

6) *Temporary structures*
   In many cases, mappings are inter-related (see e.g. the homomorphic patterns given in table 5.6), or may require intermediate computations, depend on certain globally assigned values etc. To tackle such issues, provisions for the definition of temporary data structures have to be foreseen. Such data structures should be capable to accommodate any of the basic data types needed by the language, value assignment should be as flexible as possible, and the use of variables should be aligned with the overall declarative style of the language.

7) *Initialisers*
   It may happen that the source model in a mapping contains less information than required by the target. Whilst the creation of completely new entity classes is not of primary importance for the assumed implementation strategy, the generation of attribute values and/or references not contained in the source structures has to be supported to allow a modeller, or an end-user, to fill in the gaps. It should be possible to make such specifications both with the help of functional code and interactively, during the execution of a particular mapping task.

8) *Entity selection*
   Quite often the mapping of classes can and must be done in more than one way, depending on some specific conditions. Therefore, it should be possible to specify alternative mappings for a given class, along with detailed conditions determining when to apply them. A useful feature, though probably not in the scope of the language itself, would be the ability to specify such conditions interactively, for example by selecting the source object instances from an appropriately presented visualisation (CAD or VRML views, hypertext etc.).

9) *Unit and type handling*
   Engineering data models contain a lot of physical quantities expressed in terms of measure values. Since different models may use different units of measure or even different data types for the same physical quantity, it is necessary to support unit and type conversions as well as different numeric precisions as indicated in table 5.2, item II.B.

5.7 Related Model Mapping Approaches

In the last decade several model mapping approaches have been suggested for different purposes, ranging from the tackling of model development issues, such as schema migration or schema evolution, up to incremental updates in interactive integrated environments. All these approaches tackle to some extent the model transformation problems addressed in the previous section, but none of them provides the full range of features that were identified. However, in spite of their notably different goals, they have at least two things in common:

1) Providing a method to specify the mapping between two (sets of) models by means of a formal language specification, and
2) Enabling the practical realisation of a mapping task within the covered scope of problems with the help of stand-alone tools or by modules that are part of a more sophisticated environment.

In the context of this study, especially the underlying languages of the developed mapping approaches are of interest.

5.7.1 Transformr

Historically, Transformr (Clark 1992) is the first mapping formalism developed in the product modelling arena. It has been designed for use in ISO STEP with the primary goal to enable the propagation of changes between different versions of a model. Such model transformations are achieved with the help of a relatively simple mapping language, coupled with a tool that performs the given mapping specifications.

The Transformr language is elegant and compact, designed in a procedural style. Its main constructs are COPY (responsible for the replication of source entities) and BUILD (providing the instructions for creating new entities in the target model from a set of entities in the source). Separate BUILD statements are used for each entity level mapping specification. The possibility to define conditions for the mapping of entity instances is given through a WHERE subclause.

Whilst it appears to be well suited for the purposes for which it has been designed, Transformr has several drawbacks w.r.t. the requirements of a concurrent engineering environment. It has a very limited functionality which does not allow clustering of entities, post-conditions, or associations of instances in the target. It is also quite limited in the types of equivalences that can be specified between attributes and does not support partial mappings.

5.7.2 EXPRESS-M

EXPRESS-M (Bailey 1995) has also been developed exclusively for the ISO STEP standard. Its main purpose is to facilitate application protocol interoperability.

Because of the explicit intention to be used in ISO STEP, the EXPRESS-M language is designed to have the same look and feel as EXPRESS. Its scope includes the tackling of unidirectional mappings between whole models and between model versions. Partial model updates and mapping/matching to existing models are not supported. Instances of the target model can be created explicitly, with the help of the basic language construct MAP, or implicitly, by reference from other entities. Duplicate instances that may occur by a combination of an explicit MAP definition and one or more implicit mappings can be eliminated by means of explicit PRUNE statements. An interesting feature of the language is the possibility to specify, by means of a TYPE_MAP construct, how a value of a defined type should be mapped to a value of another type. This feature can be useful for unit handling.

An EXPRESS-M mapping specification is built of both declarative and procedural components. The actual mapping is executed like program code, in the order of the mapping specification. This requires from the developer certain programming skills, makes the coding less obvious and does not allow to concentrate on the model transformation task as such.

As a whole, EXPRESS-M is focused mainly on schema integration with objectives similar to the database integration approaches described in section 5.5. It is well suited for use in the development process of large environments based on consolidated product data schemas such as the STEP APs 203 "Configuration controlled design" (ISO 10303-203 1994) and 214 "Core data for automotive design" (ISO 10303-214 1997) used in the automotive industry. Because of the lack of such stable specifications in the AEC domain, EXPRESS-M is not very appropriate for a concurrent engineering environment for building construction. Other drawbacks are the lack of an interactive regime, the complicated conditional mappings in cases where more than one transformation for a given class is needed, and the missing support for some attribute mapping patterns, such as the associative mappings requiring new object references to be generated in the target model.

5.7.3 EXPRESS-V

EXPRESS-V (Hardwick 1994) is another mapping approach closely related to ISO STEP. It has been developed as an extension to EXPRESS with the main goal to support database views in a STEP-based environment, assuming the existence of a comprehensive integrated model, such as the STEP AP 214 mentioned above. Probably for this reason, EXPRESS-V provides only language constructs that are intended for inclusion in the schema of that common integrated model, from which all other models are supposed to be derived.

The mapping specifications in EXPRESS-V are mostly procedural. Its main construct, VIEW, closely resembling similar relational database definitions, provides subclauses for attribute level mapping specifications (VIEW_ASSIGN) and for the definition of mapping conditions (WHEN). A VIEW statement can contain also certain bi-directional definitions intended to ensure the consistency between the integrated model and the derived (subordinate) view models. In particular, it is possible to specify what should happen to a source model entity instance if a respective view model instance is created, deleted or modified and vice versa. However, this feature also inter-mixes semantic and behavioural interoperability issues which decreases the modularity of the approach.

EXPRESS-V is also not quite suited for the purposes of a concurrent engineering environment for building construction. It does not support partial mappings and assumes that the source schema is always semantically richer than the target schemas to be mapped to. As a result, all mapping specifications are associated with one particular schema, many mapping patterns are neglected, and a strictly standardised naming policy for a huge name space is required. All this does not match the strongly fragmented model world of AEC.

5.7.4 EXPRESS-X

EXPRESS-X (ISO/TC184/SC4/WG11/N088 1999) is an evolving language being developed with the joined efforts of the authors of EXPRESS-M and EXPRESS-V. It aims at a more comprehensive model mapping support for STEP-based environments, combining the MAP and VIEW features of EXPRESS-M and EXPRESS-V, but retaining also some of their drawbacks, such as the basically procedural style of the mappings, the absence of provisions to describe partial mappings, and the limited support for multiple source/target associations of the entity classes in a model schema.

During the last four years there have been efforts to make EXPRESS-X part of the STEP standard. Because of this normative orientation, the main emphasis of the work has been on the language specification and not on implementation issues, although the documentation contains some useful hints for the realisation of a mapping engine. In fact, the language has been implemented successfully in a few commercial product data management environments used in the automotive and the process plant industries (EPM 1996; STEP Tools 1999), but it is also criticised in the STEP community for being complicated for broader usage, and also for not being able to handle certain practical requirements. Its acceptance, in spite of several useful revisions, is still moderate. The known examples of EXPRESS-X implementation are focused on its use in large-scale environments centred around a large standardised integrated data model.

5.7.5 XP-Rule

XP-Rule is part of the XPDI toolset. The development of this toolset has been initiated at CSTB, France (Poyet 1993) for the specification and the rapid prototyping of STEP-based product models. Besides XP-Rule, currently XPDI offers a graphical and textual user interface for the development of EXPRESS schemas, a LISP-based late binding implementation of SDAI (ISO 10303-22 1998) and possibilities for extending a model with knowledge and reasoning components.

The XP-Rule mapping language (Zarli 1995) allows the definition of bi-directional views and provides a high level of adaptability due to its declarative nature. As its name suggests, an XP-Rule mapping specification is mainly rule-based. Each set of classes to be mapped is defined with the help of a compound RULE statement with sub-clauses for attribute level equivalences (AFFECT), temporary variable definition (LET), creation of new data structures (CREATE), IF-THEN mapping conditions etc. In addition, the integration of LISP and C function code is also possible.

Though XPDI is basically a rapid prototyping platform which does not have the objectives to provide full support for product data management in complex integrated environments, XP-Rule is capable of solving many of the outlined mapping problems for such environments. It has been used successfully in the COMBI project for the partial transformation of an application-specific model to the STEP AP 201 (cf. Scherer & Sparacello 1996), as well as in the ATLAS project (Poyet et al. 1994a). However, these exercises have exposed also some deficiencies of XP-Rule, such as the inability to create complex entities and to generate attributes from entity names. The run-time performance of the mapping engine is also not very satisfactory, probably because of the adopted almost pure rule-based approach. Therefore, to the knowledge of the author, XPDI is not being further developed at CSTB.

5.7.6 Operation Mapping (OM)

OM is a commercial tool developed at Metaform Software, Amsterdam (Bijnen 1995). It includes a very powerful and efficient mapping engine and a convenient graphical user interface for specifying mappings interactively.

The underlying mapping formalism is based on relational theory. Both the source and the target data models are converted to relational database schemas on which the mapping is applied. The language used by OM is descriptive and is closely related to the data definition languages of modern RDBMS. It is capable of dealing with almost all kinds of constraints, conditions, entity and attribute equivalences that can be expressed by relational algebra constructs. Entity and attribute mappings are supported for all cardinality cases.

OM has been applied successfully for several practical tasks. In the COMBINE 2 project (Augenbroe 1995b) it has been used to derive automatically a visualisation model from the sophisticated central IDM. In another case it has been applied for automatic restructuring of the layers in a set of architectural drawings. However, since OM requires that the source and target models are translated to relational representations, it is limited to mapping tasks where such translation is possible. Another limitation of OM is the lack of interactive, partial mapping support.

5.7.7 EDM-2

EDM-2 (Eastman et al. 1995a, b) is an innovative engineering database environment developed at UCLA. It incorporates three important features not found in traditional database systems: (1) dynamic schema specification and schema evolution, (2) built-in consistency management, and (3) explicit translation definitions for mapping to/from design applications.

Mapping is not the main feature of EDM-2. It has been introduced in a later development phase, after noticing that mapping constructs are required for the integration of external applications (Assal & Eastman 1995).

All mappings in EDM-2 are specified in a descriptive manner, with the help of two high-level constructs – MAP and MAPCALL. MAP provides the *definition of the process* through which entities are translated from the source to the target model. MAPCALL describes the use of a MAP definition for particular instances in the source model. Detailed attribute equivalences are not supported by the EDM-2 language, and are implemented directly as C++ methods and functions.

EDM-2 aims to provide comprehensive support for distributed multi-discipline design environments. Although there are still some open questions, many of the addressed issues are important contributions in the area of engineering database research. However, with respect to mapping problems EDM-2 is not yet sufficiently developed. A great portion of the mapping is language dependent, there is almost no code re-use, and all attribute level mappings are invisible to the user which makes them difficult to understand and maintain.

5.7.8 ACL/KIF

ACL (Agent Communication Language) supplemented by KIF (Knowledge Interchange Format) has been proposed in the DARPA knowledge sharing initiative as a method for exchanging information between intelligent interactive agents with the main objective to enable the co-operation of knowledge-based design tools (Genesereth & Fikes 1992).

ACL/KIF presents a principally different approach for tackling the semantic interoperability problems in a distributed modelling environment. It follows a messaging paradigm based on a blackboard system architecture, and is able to handle the following two types of problems (Khedro et al. 1994): *vocabulary translation* (mapping) and *logical translation* (conversion of messages from the native format of the sender to the local format of the receiver). Mappings with ACL/KIF can be bi-directional or unidirectional, partial or complete. Along with mapping, the propagation of changes and the continuous consistency of the data in all distributed agents are considered as well. All these features are wrapped in the same messaging format which makes it somewhat convoluted and difficult to implement.

In principle, ACL/KIF provides a very high-level of support for co-operative design work. However, it demands sophisticated agent functionality by obliging each agent to be aware of all shared modelling concepts, and to take on the responsibility to perform the necessary transformations by itself. Such requirements are not practical when existing "conventional" applications need to be considered. Some other drawbacks are the relatively low modularity of the approach, the missing consideration of STEP/EXPRESS, and the strong coupling with a specific system architecture.

5.7.9 VML

The View Mapping Language (VML) has been developed as part of a Ph.D. thesis with objectives that are closely related to the needs of a concurrent engineering environment (Amor & Hosking 1995; Amor 1997).

VML provides detailed constructs for most of the mapping problems raised in the previous section. It is principally designed to support bi-directional mappings and, consequently, makes no clear distinction between source and target objects. Although not uncomplicated, the language syntax is well structured, following a strictly declarative style. Each mapping specification begins with an INTER_SCHEMA declaration in which the regime of the mapping is fixed. In particular, details are given if the mapping will be partial or complete and what should be the type of access to the involved models (READ_ONLY, READ_WRITE, INTERACTIVE). Each class level mapping is described by an INTER_CLASS declaration providing constructs for detailed attribute mapping definitions (through the subclause EQUIVALENCES), conditions for determining the set of instances to be mapped (through the subclause INVARIANTS), and pre-setting of initial and/or default values (through the subclause INITIALISERS).

VML has been conceived in first place for the solution of the mapping problems in integrated design environments. In the prototyped software implementation, the language is supplemented with a graphical interface for visual development, and a mapping engine which performs not only mapping, but also change management and version control tasks. However, all these features are centred around a prescribed integration approach (assuming a database-like environment with a common shared integrated model) which has influenced to some extent the design of the language itself. Some minor drawbacks of VML include the limited built-in features for user interaction and the lack of micro-level control on attribute mappings which can make some of the attribute mapping patterns difficult to represent. More concerns provides the VML-based engine with the intermixing of mapping and matching methods, as well as some automated decisions which are difficult to accept. The whole process is not sufficiently transparent and relies too much on the representational power of the language, leaving little room for corrective actions.

5.8 Proposed Approach for Tackling the Semantic Interoperability in CEE

In summary, an appropriate approach for tackling the semantic interoperability in CEE should:

1) enable the solution of the problems identified previously in this chapter, and
2) fit seamlessly into the suggested overall environment.

Though not evident at first glance, these two issues are of almost equal importance. A solution that is not aligned with the other components of the environment would be yet another "island of automation" and thus fail to achieve the envisioned high-level goals.

From the analysis of all presented aspects, coupled with the concurrent engineering requirements to product data management and the general concepts of the modelling environment outlined in chapter 3, a modular model mapping approach, broken down into the following development steps, has been synthesised:

1) Design of a mapping language in accordance with the requirements from section 5.6.5;
2) Development of appropriate implementation methods for the realisation of a mapping engine which should act as a *server-side agent program* performing all model transformation tasks requested by client applications;
3) Specification and implementation of public high-level operations enabling the use of the mapping services when and as necessary, and aligned with all other services offered by the environment;
4) Providing the necessary links to the components of the server platform responsible for tackling the integrity problems that have been intentionally left out of scope to isolate and reduce the requirements to the mapping task. Such components include: model matching, consistency checking, model merging, conflict management etc.

In this approach, a major role is assigned to the **mapping language**. It must enable generic, platform-independent definition of all necessary mapping models, in the same way as EXPRESS provides for the platform-independent representation of the project data models in the concurrent engineering environment.

The **implementation methods** are made responsible for: (1) parsing a mapping specification, (2) creating the *mapping domains and extents* of all affected entities[*)], and (3) performing the actual transformations of the data, including the generation of all inter-object relationships in the target model(s) and the removal of redundant data after the mapping.

The **mapping engine** should encompass all implementation methods and interact with the project data server, using the available server operations to access the needed mapping specifications, as well as the respective model schemas and model data.[**)]

[*)] Mapping domains and extents are discussed in chapters 6 and 8, along with the other components of the developed mapping system.

[**)] The mapping engine provided in the prototype realisation of the project data server outlined in chapter 8 has been written in Common LISP, as part of the server environment built on top of KEE (Intellicorp 1994). However, a detailed description of the developed algorithms goes beyond the scope of the thesis. Besides that, these algorithms need considerable improvements w.r.t. their computational efficiency which has not been the goal of the work, and is not the primary competence of the author. Therefore, in chapter 8 only the basic principles of the mapping engine are presented; more detailed coverage of the related problems is a subject for future work.

The **public high-level operations** should correspond to the general Information Container based format of all services of the environment. They are responsible to package the client requests, schedule and trigger the mapping processes, format appropriately the results and return them to the application or user that has issued the respective request.

In addition, in order to maintain a consistent link to the other relevant interoperability components, the *model/object version status* has to be appropriately updated after each model transformation operation, so that a later service, e.g. matching, can reason properly about the requested actions. This issue is of great importance because of the suggested bipartite treatment of the semantic interoperability and semantic integrity problems.

For a specific CEE, the proposed mapping system can be used as follows:

1) Represent all potentially needed model transformations as mapping models specified in terms of the developed mapping language;
2) Embed as appropriate remote procedure calls in the client applications to invoke the respective server operations initiating a mapping; alternatively, provide a client adapter to access the project data server, along with a respective local interface to transfer the data from/to that adapter and the application program;
3) Use whenever necessary the supported operations to coordinate and synchronise the domain models processed locally by the individual professionals involved in a project.

The first two of the above steps have to be accomplished *before* the environment is used in a specific project. However, this must not be done at the initial design phase of the CEE system, because a gradual enhancement, including the definition of new models and new mappings, does not affect the generic methods of the project data server, and requires only minimal adaptation of specific methods and existing client applications.

Fig. 5.5 below illustrates the principal interaction of the mapping module with the other components of the suggested environment.

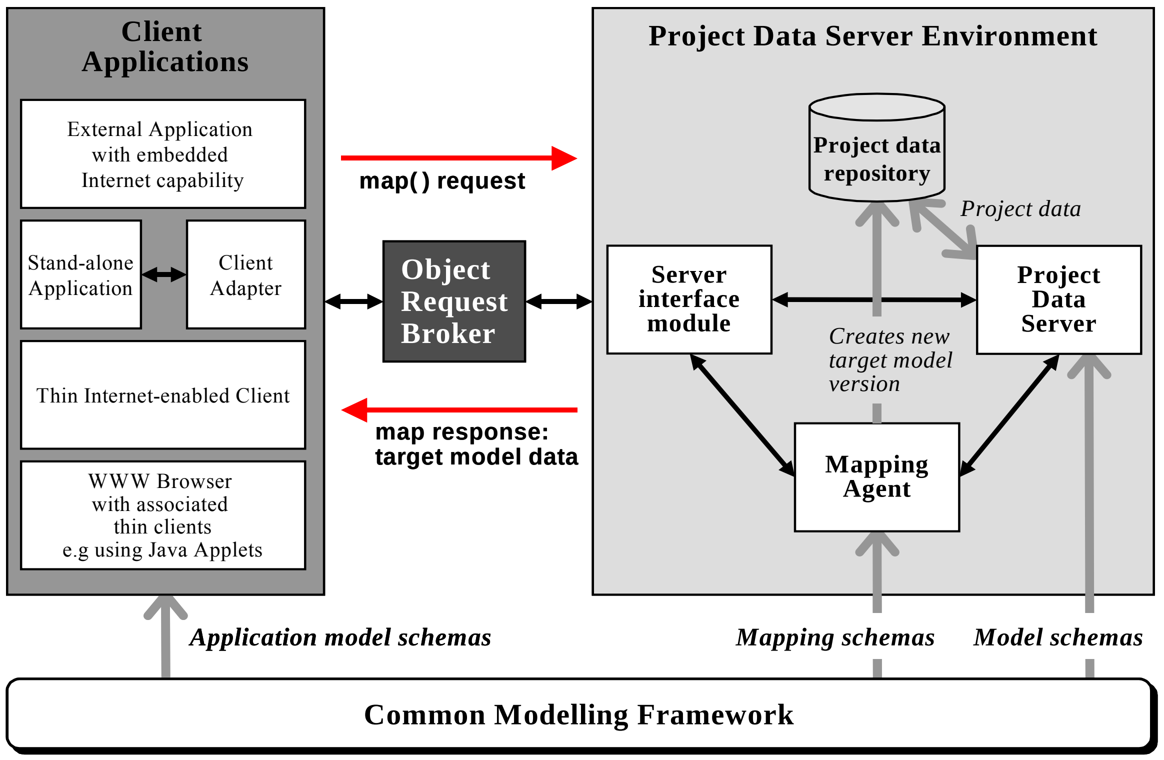

Client Applications
External Application with embedded Internet capability
Stand-alone Application
Client Adapter
Thin Internet-enabled Client
WWW Browser with associated thin clients e.g using Java Applets
map() request
Object Request Broker
map response: target model data
Project Data Server Environment
Project data repository
Project data
Server interface module
Project Data Server
Creates new target model version
Mapping Agent
Application model schemas
Mapping schemas
Model schemas
Common Modelling Framework

Fig. 5.5: Principal schema of the use of the mapping system in the proposed CEE

Fig. 5.6 schematically shows the suggested interplay of model mapping, matching and merging in a running project. In this figure, representing one design interaction cycle, at the starting point T_0 the domain-specific models used by the architect and the structural engineer are assumed to be consistent, where the latter is derived through a mapping operation from the architectural model, i.e. $A0 \rightarrow S0_{map}(A0)$. Later on, in the process of concurrent work, these models inevitably diverge. However, by mapping the modified architectural model $A1$ to obtain a structural model $S1_{map}(A1)$, and afterwards matching that model with the local structural model $S1_{local}$, the two domain models can again be brought in a consistent state at point T_1 within a cooperative coordination session[*)].

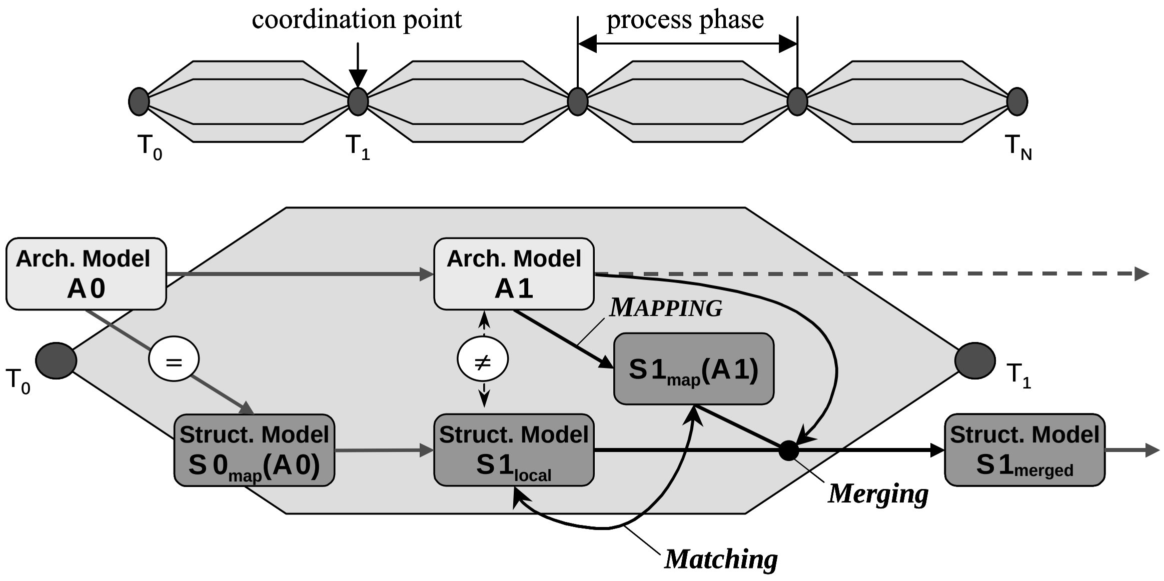


Fig. 5.6: Schematic presentation of the interaction between the mapping, matching and merging services in the project coordination process

In the next chapter 6 the developed mapping language (CSML) supporting the outlined approach is described, and in chapter 7 selected examples of model transformations are given. The prototype realisation of a project data server providing the interaction of all developed components is shown in chapter 8.

5.9 Discussion

The problems of semantic interoperability are well recognised in database research to date. There exists a lot of literature on the topic, where a number of different approaches for the integration of mulitdatabase and federated database systems can be found. However, the developed methods in this domain are suitable mostly for business databases with relatively simple structure, whereas object-oriented models, typical for the engineering domain, are yet poorly supported. This is only partly due to the relatively weak interest in engineering database solutions on the side of computer scientists in general. The deeper reasons for the

[*)] In the given example, it is assumed that the structural engineer is required to accept the changes of the architect and modify his/her data accordingly.

lack of adequate methods for tackling the semantic interoperability problems in engineering environments are that:

1) object-oriented models are more complex than relational models, with more sophisticated structure, deep specialisation hierarchies and different kinds of associations, including a lot of *has-part* aggregations, which are more difficult to deal with, and
2) the set up requirements w.r.t. data consistency and data integrity are very hard, due to the specific implementation objectives associated with the target domain of business-oriented DBMS.

Therefore, more pragmatic efforts have been undertaken in the last years in the specific area of product data management. Whilst lacking to some extent a strong theoretical basis, the various proposed model mapping approaches have been developed in view of certain practical requirements and application cases, and are well suited for the range of tasks for which they were conceived. However, with the exception of ACL/KIF and VML, all known mapping approaches do not address sufficiently the problems of cooperative and concurrent work, and ACL/KIF and VML are strongly tied to a particular environment architecture and component behaviour. In each case, a global integrated model is assumed as a prerequisite of the environment, and for that model similar requirements as in the more general database integration approaches are raised. All this has influenced the scope of the examined issues, and has led to several limitations in the developed solutions.

In contrast, the model mapping approach proposed in this thesis aims at supporting a project environment, where the data are processed autonomously and simultaneously by the separate project players. Data consistency is not guaranteed at all times, but can be provided through user-driven actions at selected points of the project progress, assisted by appropriate mapping models and tools. Thus, in terms of database technology, the proposed approach enables design, communication and execution autonomy at the expense of soft integrity requirements and renounced use of global applications, with simultaneous access to more than one domain-specific models. However, such applications are seldom needed in AEC. Global applications for project management and monitoring would normally use a higher level of data abstraction which can be defined in an appropriate domain model for "project management"[*)] or even at the system ontology level. As the needed data is expected to be of relatively limited scope, on-line mapping to/from other domain models does not seem difficult to achieve.

The differences in the objectives compared to other known approaches allow to address a broader range of mapping problems associated with object-oriented data models. Greatest attention can be paid to EXPRESS-based models in accordance with the overall goal of the work. The identified mapping types and the developed mapping patterns enable a more comprehensive treatment of the model transformation tasks that may occur in an integrated environment for concurrent engineering.

These capabilities of the proposed approach are underpinned by the qualitative comparison with the examined database integration methods given in table 5.8 below. The estimated characteristics of the different methods, except for the model mapping approach proposed herein, are compiled from a similar investigation conducted in (Conrad 1997). The features characterising the different approaches are divided in three categories: the ability to support

[*)] In fact, such domain model already exists in the current IFC specification.

different modelling concepts, the level of consideration of the identified inter-model conflicts, and general considerations w.r.t. the implementation environment, as proposed in (Batini et al. 1986).

Table 5.8: Comparison of different approaches for tackling the semantic interoperability problems in non homogeneous modelling environments

Features	**Integration approaches**			**Proposed model mapping approach**
	Superviews	Assertion based integration	Formalised object-oriented integration	
I. Support of different modelling concepts				
Entity classes	High	High	High	High
Value attributes	High	High	High	High
Reference attributes	Medium	High	Low	High
Inheritance	Low	–	High	High
Object aggregations	Low	–	–	Medium
II. Consideration of different modelling conflicts				
Semantic conflicts	Medium	–	Medium	Medium
Descriptive conflicts	–	–	Medium	Medium
Structural conflicts	Medium	Medium	Low	High
Heterogeneity conflicts	Low	High	High	–
III. Consideration of basic requirements to the implementation environment				
Completeness	not proven	not proven	not proven	not proven
Correctness	Yes	Yes	Yes	Yes
Non redundancy	No	No	Yes	No
Understandability of the integrated model	Good	Good	Good	not an issue
Guaranteed continuous consistency	Yes	Yes	Yes	No

By examining this table, the different goals of the separate methods become more evident. The support of a larger number of modelling concepts and the consideration of structural conflicts are better provided by the model mapping approach, whereas such issues as the consideration of heterogeneity conflicts, the understandability of the overall model, and the requirements for non redundancy and continuous consistency of the data are not in its scope.

Here it should also be noted that the *completeness* of all methods is marked as “non proven”. This is due to the fact that the mathematical foundation of object-oriented modelling lies in higher-order logic. As proven by Kurt Gödel, whilst first-order predicate logic is complete and consistent, but not definite, higher-order logic is neither complete, nor definite (Gödel 1931). Thus, an attempt to identify all possible inter-relationships between the entities of two object-oriented models on the basic of the generic mapping types specified in section 5.6.3 can lead to a combinatorial explosion when the full set of relations for objects with arbitrary number of attributes is tried to be determined. However, the identified mapping patterns, drawn from the study of existing and envisaged practical cases, broaden the scope of supported issues and give greater conceptual clarity both for the design of a mapping language and for the realisation of many real-world tasks.

A final point to discuss is the capability for bi-directional mappings. Whilst this can be of great benefit for the design and maintenance of an integrated environment, unlike some other researchers, I disbelieve its practical value. The main reason for that is, again, the complexity and diversity of the problems associated with the possible inter-relationships between the entities of sophisticated models based on a higher-order logic paradigm.

Nevertheless, in practical cases where the data transformations between highly harmonised models are the primary goal, as e.g. in the IFC framework, a limited capability for bi-directional mapping is worth considering.

Further work looking at requirements for such cases can provide an improvement to the overall treatment of the addressed semantic interoperability problems.

Chapter 6: The CSML Mapping Language

We must either institute conventional forms of expression or else pretend that we have nothing to express.
– George Santayana, Soliloquies in England

With the Context-Independent Schema Mapping Language (CSML), developed as main component of this thesis, many of the problems identified in the previous chapter can be overcome. This chapter presents the basic concepts that have governed the design of CSML, outlines its overall structure and its principal components and details its formal syntax. To prove the functionality and the implementation potential of the language, the formalisation of the developed mapping patterns, the representation of basic relational algebra operations and the solution of typical mapping exercises drawn from selected literature sources are examined. The presentation concludes with an appraisal of CSML with respect to the specified objectives.

6.1 Basic Concepts

Unlike most other examined approaches that focus mainly on schema evolution and schema migration, or on the generation of database views, implicitly presuming the existence of one shared global model in the environment, CSML has been conceived for run-time interoperability support in a distributed, non harmonised modelling environment, rather than as a model developing aid or SQL-like view generation utility. The emphasis in CSML is on the data-level transformations between (any) populated project models. It can be applied to a variety of model transformation problems, no matter if the involved models are globally used, shared by many project actors, locally owned by one actor or dedicated to serve specific applications.

The **role of CSML** in the proposed approach for the solution of the interoperability problems in CEE through a series of autonomous operations (mapping, matching, consistency checking, merging, conflict resolution) is *to enable the transformation of the data* contained in one data model (source) to their appropriate representation according to the conceptual schema of another model (target), without being aware of the context, i.e. the locally stored data in that target model. This context-independent approach to the mapping problem considerably simplifies both the design of the language and the implementation of an appropriate mapping engine, and at the same time increases its representational capacity.

In accordance with the requirements suggested in chapter 5, CSML has been designed in pronouncedly **structured, declarative style**. All language constructs are logically inter-connected, but the sequence in which the individual mappings should be performed is neither prescribed, nor enforced in any other way. Thus, in effect, a mapping specification in CSML declares, but does not prescribe a model transformation. The sequence of execution of the entity/attribute mappings is governed only by the context of the specific problem and is determined at run-time by the mapping engine.

To get a first impression of CSML, let us consider a small example problem which should be intuitively clear. It presents a small, adapted portion of the second demonstration example outlined in chapter 1. The task is to map the data from a populated IFC project model (heavily simplified here) to a dedicated (also simplified) model for structural design applications.

Source schema:

```
SCHEMA SimplifiedIfcProjectModel;
ENTITY IfcRoot;
  label : STRING;
END_ENTITY;
ENTITY IfcElement
  SUBTYPE OF (IfcRoot);
  ElementType : STRING;
END_ENTITY;
ENTITY IfcBuildingElement
  SUBTYPE OF (IfcElement);
  HasMaterial : OPTIONAL STRING;
END_ENTITY;
ENTITY IfcRelAssemblesElements
  SUBTYPE OF (IfcRoot);
  RelatedElements :
      LIST [1:?] OF IfcElement;
  RelatingElement : IfcElement;
END_ENTITY;
...
END_SCHEMA;
```

Target schema:

```
SCHEMA SimplifiedStructuralModel;
ENTITY StrComponent
  ABSTRACT SUPERTYPE OF
    ONEOF(StrAssembly,StrElement);
  name : STRING;
END_ENTITY;
ENTITY StrAssembly
  SUBTYPE OF (StrComponent);
  has_elements :
      LIST [1:?] OF StrElement;
END_ENTITY;
ENTITY StrElement
  SUBTYPE OF (StrComponent);
  material_name : STRING;
  material_properties :
      LIST [1:?] OF NUMBER;
END_ENTITY;
...
END_SCHEMA;
```

Mapping specification in CSML:

```
(MAP SimplifiedStructuralModel FROM SimplifiedIfcProjectModel
 COMMENTS "Mapping of shared project data as input to structural design"
 PRESETS (LOAD "utility-fns.lisp" FOR MaterialLookUp)
 CLASSES
  (MAP CLASS StrComponent FROM IfcRoot
    ATTRIBUTES (SAME name AS label))
  (MAP CLASS StrElement FROM IfcBuildingElement
    CONDITIONS (ElementType = "structural")
    ATTRIBUTES
      (SAME material_name AS HasMaterial)
      (MAKE material_properties
            CONSTRUCTOR MaterialLookUp ARGS HasMaterial))
  (MAP CLASS StrAssembly FROM IfcRelAssemblesElements
    VAR (MAKE ?RelVar FROM RelatingElement)
    CONDITIONS (?RelVar -> ElementType = "structural")
    ATTRIBUTES
      (MAKE has_elements FROM RelatedElements)
      (MAKE name FROM ?RelVar -> label))
 )
```

Fig. 6.1: Example mapping with CSML

For this, somewhat artificial example, if the source model data (in STEP file format) are:

```
#10=IfcBuildingElement('Frame_1','structural',$);
#11=IfcBuildingElement('Beam_1','structural','C25');
#12=IfcBuildingElement('Column_1','structural','C35');
#13=IfcBuildingElement('Column_2','structural','C35');
#14=IfcBuildingElement('Wall_13','architectural','Masonry');
#20=IfcRelAssemblesElements('Assembly_1',(#11,#12,#13),#10);
```

the target model data, after mapping and exporting them as a STEP physical file, could for example be:

```
#1=StrElement('Beam_1','C25',(29000.,20.,2.2,-3.4,-3.5));
#2=StrElement('Column_1','C35',(32000.,30.,2.9,-3.2,-3.5));
#3=StrElement('Column_2','C35',(32000.,30.,2.9,-3.2,-3.5));
#4=StrAssembly('Frame_1',(#1,#2,#3));
```

where the material properties for `C25` and `C35` (E_{cm}, f_{ck}, f_{ctm}, ε_{cu}, $\varepsilon_{cu,s}$ etc.) are created by the external function `MaterialLookUp`, which is assumed to find a material by name in a table of pre-defined materials and return a list of its properties in a preset order.

Each model mapping specification in CSML consists of a set of nested **mapping declarations**. Each such declaration starts with a *left parenthesis*, may include several detailed *specification clauses*, as well as other enclosed mapping declarations, and ends with a *right parenthesis*. The example in 193 above consists of one single schema mapping declaration (`MAP SimplifiedStructuralModel` …) which encloses 3 class mapping declarations (`MAP CLASS`) with several subclauses, containing more specific lower level declarations. This setup strongly resembles the structure of a typical LISP routine.

In order to access the elements of the involved schemas, the following **named data objects** are used:

```
schema_name  =  symbol .
class        =  [ schema_name ':' ] symbol | variable .
attr         =  [ class '->' ] symbol | variable .
variable     =  '?' symbol .
keyword      =  [ '_' ] symbol .
symbol       =  letter { letter | digit | '-' | '_' }* .
```

Schema, class and attribute names must correspond to the respective source/target schema definitions. Variables can be defined globally (for the whole mapping) as well as locally (for the scope of one declaration). They do not have fixed data types, but are bound to particular types (real, integer, string, object etc.) at their initialisation.

Keywords, such as the words `MAP`, `FROM`, `COMMENTS`, `PRESETS`, `CLASSES` in 193, are basic language tokens that must be written exactly as indicated. Whilst CSML is actually not case-sensitive, in 193 and in all further examples and syntax definitions keywords are given always in capital letters for emphasis*).

To be able to work with the data, it is also necessary to have a set of **operators**. Unlike most other languages, CSML concentrates exclusively on operations directly related to the mapping equivalences and therefore defines – intentionally – a very small set of only three types of operators: 5 prefix *term operators* (`ONEOF`, `LISTOF`, `SETOF`, `MAX`, `MIN`) enabling the construction of complex terms, 7 *relational operators* (`= < > <= >= <> :=:`) enabling the comparison of different data objects, and 1 *addressing operator* (`->`) used to access embedded elements in complex structures. For arithmetic operations (if at all needed) pure Common LISP code is adopted (see Steele 1990).

These, in short, are all the basic elements of CSML. In the next section the different types of mapping declarations and their component clauses are discussed in more detail.

*) The keywords used in the syntax boxes in section 6.2 are not enclosed in apostrophes for brevity. The formal definition of all 58 keywords of CSML is presented in section 6.3 (rules 9 to 66).

6.2 Structure and Components

In CSML each mapping specification is arranged in a strictly hierarchical manner as outlined in fig. 6.2 below. This structuring is applied consistently to all mapping problems, regardless of the specific data models being dealt with.

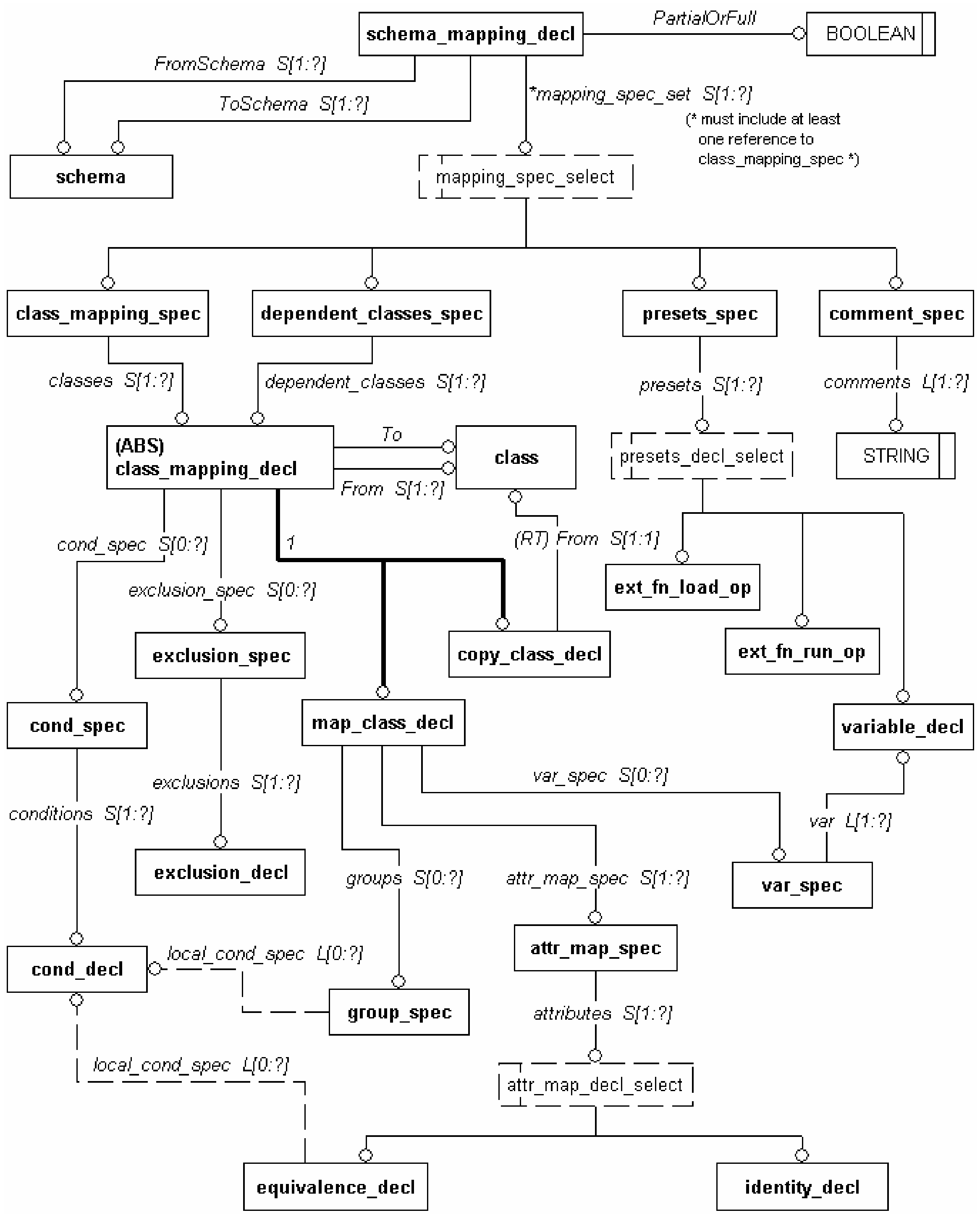


Fig. 6.2: Overview of the high-level constructs of CSML in EXPRESS-G

6.2.1 Top-level constructs

A full model mapping specification is comprised effectively of just *one single statement*, **`schema_mapping_decl`**, which encompasses all detailed mapping declarations on class and attribute level. Its syntax is as follows:

```
schema_mapping_decl    = '(' MAP [ PARTIALLY ] { schema }+
                               FROM { schema }+
                               mapping_spec_set ')' .
schema                 = [ SHARED ] schema_name .
```

For example:

```
(MAP vrml_view_model FROM SHARED IfcProjectModel
 <more detailed mapping declarations> ...
 )
```

Normally, each data model supported in the modelling environment will be described by one conceptual schema. CSML accepts also the specification of multiple schemas for consistence with the short form schema specification defined in ISO STEP which allows references to external schemas, such as the STEP resources (ISO 10303, parts 41 to 49).

A schema can be mapped *fully*, meaning that the mapping operation should create a complete, consistent target model, or *partially*, meaning that only some instances of the target model will be created, and not all values and reference attributes corresponding to the class declarations will necessarily be populated. CSML assumes full mapping by default; partial mappings are specified through the keyword `PARTIALLY`. In fact, this information is not needed by the mapping engine itself, but to mark the status of the created target model which can be important for subsequent operations on that model. For similar reasons, the keyword `SHARED` is included, to denote if the model to be mapped from/to is shared or used locally, i.e. in a private workspace. It has only informative meaning to the mapping itself.

The detailed mapping declarations, encompassed in a `mapping_spec_set` are organised in four sections:

```
mapping_spec_set       = { { class_mapping_spec }+ |
                               dependent_classes_spec |
                               presets_spec | comments_spec }+ .
where:
class_mapping_spec     = CLASSES { class_mapping_decl }+ .
dependent_classes_spec = DEPENDENTS { class_mapping_decl }+ .
presets_spec           = PRESETS { presets_decl }+ .
comments_spec          = COMMENTS { string }+ .
```

Each of these sections can be repeated 0 or more times in arbitrary order. However, at least one `class_mapping_spec` must exist in order that the mapping makes sense.

The **`class_mapping_spec`** sections contain the primary inter-class mapping declarations, instructing the mapping engine how the object instances of the source model are to be transformed into object instances of the target model. These instructions are defined generically on class level, but are executed on instance level, depending on the specified conditions and the given source model context.

The **`dependent_classes_spec`** is a feature not found in other mapping languages. Structurally it is almost identical to `class_mapping_spec`, but its purpose is different.

Whilst the mappings declared in a `class_mapping_spec` section are mandatory and are therefore performed for all instances of the source model classes (constrained only by the eventual specification of additional selection conditions), the instances addressed in a `dependent_classes_spec` section are mapped to the target model if and only if they are referenced by "primary" object instances in such mandatory declarations. This avoids the mapping of typical resource instances, such as `IfcPoint`, `IfcMaterial`, `IfcCost` or `IfcDateAndTime` in the IFC Project Model, if they are not needed by a "primary" object instance in the target model. For example, there may exist many `IfcMaterial` entities in an architectural domain model, defining the material properties of building objects like `IfcWindow`, `IfcDoor` etc. which need not be mapped to a structural domain model. However, `IfcMaterial` *must* be mapped on class level because it is used in "structural" objects like `IfcColumn` or `IfcWall` as well. Instead of specifying a complicated exclusion condition examining the types of objects referencing `IfcMaterial`, in CSML it is sufficient to include the mapping for `IfcMaterial` in the `dependent_classes_spec` section, instructing the mapping engine to perform the mapping only for `IfcMaterial` instances associated with the relevant "primary" objects, such as `IfcColumn` and `IfcWall`, but not `IfcWindow` and `IfcDoor`.

The remaining two sections of `class_mapping_spec` have supporting functions. The `comments_spec` section includes merely a list of strings serving as a simple documentation aid. The `presets_spec` section enables the declaration of *global variables* and *external functions* that can be used at any other place in the mapping specification.

```
presets_decl  = ext_fn_load_op | ext_fn_run_op | variable_decl .
```

Global variables are used to store intermediate results that would allow to simplify certain mapping definitions, or to reuse data from one mapping definition in one or more other definitions. They are described in more detail in section 6.2.4 below.

External functions are useful in cases where the declarative style of CSML is not sufficient to cover functional transformations with pronounced algorithmic character. How such functions can be imported and used in a mapping specification is outlined in section 6.2.6.

6.2.2 Class mapping constructs

A mapping between classes can be specified in two possible ways: by a `copy_class_decl` or by a `map_class_decl`.

```
class_mapping_decl  = copy_class_decl | map_class_decl .
copy_class_decl     = '(' COPY [ CLASS ] class [ FROM class ]
                          { cond_spec | exclusion_spec }* ')' .
map_class_decl      = '(' MAP [ CLASS ] class FROM { class }+
                          { { attr_map_spec }+ |
                            group_spec |
                            var_spec |
                            cond_spec |
                            exclusion_spec }+ ')' .
```

These two declarations have many things in common. Both define the classes participating in the mapping, an optional set of *conditions* allowing to reduce the number of instances selected for the mapping, and an optional set of *exclusion specifications* that can be used to override the default inheritance of the mappings provided for superclasses of the source classes. In addition, a `map_class_decl` may include also the *initialisation* of local variables (`var_spec`) as well as instructions to *group* the source entities in a list before the mapping (`group_spec`), and must include at least one *attribute mapping specification*.

In principle, both class mapping constructs have the same goal. However, for simple equivalences where all attributes of a source entity must be copied "as is" to the target, **`COPY CLASS`** provides a much simpler and easier to use syntax. It has been introduced to take into account the potential existence of many such mappings when pre-harmonised models are involved, as foreseen e.g. in the IFC architecture. For such models, the prevailing part of a mapping specification will contain simple declarations like:

```
   (COPY CLASS IfcBuildingElement)
```

eliminating the necessity to specify all class attributes (which can be quite a long list).

The **`MAP CLASS`** declaration is more complex. It contains several subclauses, introduced through the keywords `CONDITIONS`, `EXCLUSIONS`, `GROUPS`, `ATTRIBUTES` and `VAR`, which allow greater level of control over each particular mapping. The first three of these clauses are detailed below, the other two are discussed in sections 6.2.3 and 6.2.4.

The **`CONDITIONS`** specification has the following syntax:

```
cond_spec      = CONDITIONS [ ALL | ONEOF ] { cond_decl }+ .
cond_decl      = '(' term rel_op term ')' |
                 '(' PRED { fn_ref | KB_Template } ')' .
```

As mentioned, it serves the purpose to reduce the set of entity instances which are candidates for the mapping. The keywords `ALL` (the default) and `ONEOF` enable an overall level of control by specifying if all or just one of the conditions has to be fulfilled in order to include a specific instance in the set of instances to be mapped.
A `CONDITIONS` specification may itself contain one or more *condition declarations*. Each such declaration defines a predicate that has to be fulfilled for the examined entity instance to pass the test. There are three possibilities to specify such predicates: a direct comparison of two terms by means of a relational operator, an external or in-line Common LISP function, and an embedded knowledge-based template as defined in section 4.7.

As a complementary example to 193, assume that only the mapping of columns with height h > 25 cm is needed. The condition for this case would then be given as:

```
   CONDITIONS (h > 30)
```

If, in addition, a constraint on the area of the cross section of rectangular columns is required, e.g. A > 400 cm^2, involving a combination of more than one attribute of the "column" entity, the following conditions can be specified:

```
   CONDITIONS (SectionType = "rectangular")
              (h > 30)
              (PRED (LAMBDA (X Y) (> (* X Y) 400.0)) ARGS b h)
```

In this example, it is assumed that the "column" class defines the attributes `SectionType`, `b` and `h`, but not `A`.

The **`EXCLUSIONS`** specification allows to *shadow the inheritance* of mappings from superclasses of a given source class.

```
exclusion_spec  = EXCLUSIONS { exclusion_decl }+ .
exclusion_decl  = '(' class [ FOR { attr }+ ] ')' .
```

Normally, inheritance is turned on which means that each source class in a particular mapping declaration absorbs also the mappings defined for its superclasses, paralleling the structure of the object-oriented model. CSML expects this to be the desired default behaviour because in most practical cases the structure of the target model is supposed to have considerable similarity with the source. The `EXCLUSIONS` specification provides the means to specify exceptions to that general rule. Another possible use of the `EXCLUSIONS` specification is as *simplifier*, allowing to reduce the complexity of a mapping when more than one mapping declarations for a superclass exist, with respectively different applied conditions*).

Finally, the **`GROUPS`** clause provides means to collect the instances of a class in a list and assign this to a variable.

```
group_spec       = GROUPS { group_decl }+ .
group_decl       = '(' { class }+ FOR variable
                        [ local_cond_spec ] ')' .
local_cond_spec  = WHEN [ ALL | ONEOF ] { cond_decl }+ .
```

If no conditions are defined, all instances of the specified class will be grouped. The `local_cond_spec` clause, used also in several other constructs, enables the grouping only of selected instances. Its syntax and meaning is very similar to the class-level `cond_spec` shown above.

6.2.3 Attribute mapping constructs

Attribute mappings present the bulk of definitions in a mapping specification. They contain the details covering the different mapping patterns that may occur in a model transformation. Similar to the class mapping declarations, CSML provides two forms for the declaration of attribute mappings: one for simple cases (`identity_decl`), and one for the more sophisticated equivalence types that need to be defined (`equivalence_decl`).

```
attr_map_spec    = ATTRIBUTES { attr_map_decl }+ .
attr_map_decl    = identity_decl | equivalence_decl .
```

Fig. 6.3 below presents schematically the general structure of an **`identity_decl`**.

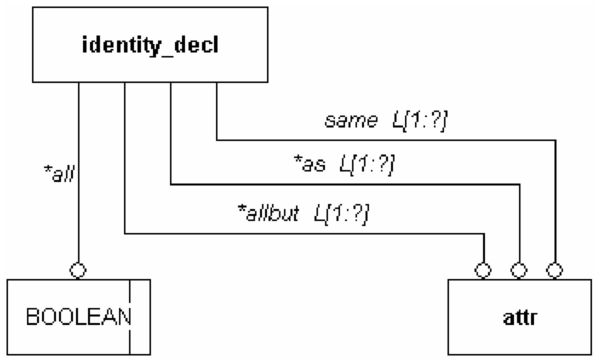


Fig. 6.3: Structure of the attribute identity declaration in EXPRESS-G

*) Without the use of `EXCLUSIONS`, it may be extremely difficult (and time consuming) to determine the set of instances to be mapped in such cases, as this would require to examine all possible combinations of candidate mappings from all relevant (super)class declarations.

Its syntax is as follows:

```
identity_decl    = '(' SAME { { attr }+ [ AS { attr }+ ] |
                               ALL | ALLBUT { attr }+ } ')' .
```

`Identity_decl` allows to specify in concise form attributes that need to be copied from the source to the target, including the possibility of renaming. For example, the declaration

`(SAME name AS label)`

from 193 instructs the mapping engine to copy the value of `label` from each relevant source instance to the attribute `name` in the respective target instance. The data type of the attribute is thereby irrelevant – it can be an integer, real, list or set of strings and so on.

For even shorter notation, it is possible to specify several such attributes in a single declaration, as well as to declare only the attributes that should not be copied from the source to the target, implying an identity transformation for all the rest. The rationale for introducing such short-form attribute declarations is the same as for `COPY CLASS`.

The general structure of an **`equivalence_decl`** is given schematically in fig. 6.4 below.

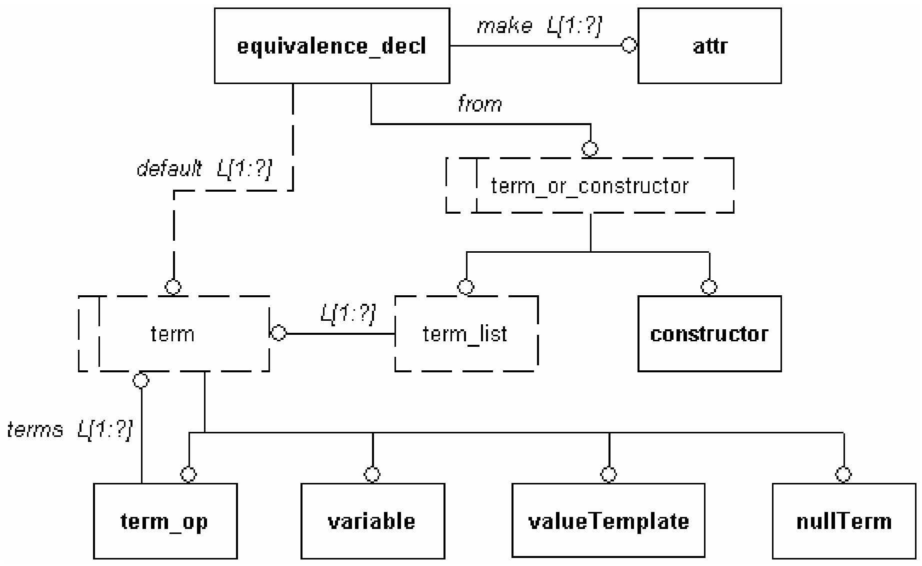


Fig. 6.4: Structure of the attribute equivalence declaration in EXPRESS-G

Expectedly, it has a more complicated syntax as follows:

```
equivalence_decl = '(' MAKE { attr }+
                       { FROM { term }+ [ suffix_op ] |
                         CONSTRUCTOR fn_ref }
                       [ DEFAULT { term }+ ] ')' .
```

Each `equivalence_decl` starts with the keyword `MAKE` followed either by a `FROM` clause, identifying the source of the mapping, or by a `CONSTRUCTOR` clause, defining a function to perform the mapping which may contain arguments referencing one or more source data. In addition, a `DEFAULT` clause allows to assign a value to the referenced target attribute(s) for the case when no value would have been determined by the mapping engine.

Whilst the target of an `equivalence_decl` must always be an attribute (referenced by name), the source data can be described in several different ways with the help of *terms*.

6.2.4 Terms and variables

Terms provide a generalised form of referencing the data in a model. They are defined in CSML as follows:

```
term             = valueTemplate | variable |
                   '(' term_op { term }+ ')' | NUL .
```

In this definition, the concept of *templates* is introduced for the first time.

In principle, *value templates* represent data from the source model which are referenced directly or as a result of template operators, lisp expressions or knowledge-based search expressions. They are discussed in more detail in section 6.2.5 below.

Variables (recognised by a starting `'?'` character) provide another method of referencing. When used as a term, the value of a variable is returned at the place where it is referenced. Of course, this requires that the variable is initialised in a respective declaration – either globally, in the `PRESETS` section of the schema mapping declaration, or locally, in the `VAR` section of the current class mapping declaration defining its scope.

```
var_spec         = VAR { variable_decl }+ .
variable_decl    = '(' MAKE { variable }+
                       { FROM { term }+ [ suffix_op ] |
                         CONSTRUCTOR fn_ref } ')' .
variable         = '?' symbol .
```

It can easily be seen that the declaration of variables is almost identical in form to the declaration of attribute equivalences. The only differences are that after `MAKE` the name of a variable, and not the name of an attribute has to be specified, and that there is no `DEFAULT` value assignment as it would not be meaningful here. Variables are not declared with fixed data types and therefore can “absorb” any data returned by the term(s) in the `FROM` clause or by the `CONSTRUCTOR` function. Thus, if a variable is assigned a complex data structure, it can later be used in the same way as that data structure. In particular, if a variable is bound to an entity, it is possible to reference that entity’s attributes by using the addressing operator `'->'`. An example of such usage is given in 193 for the variable `?RelVar`, i.e.: `(?RelVar -> ElementType = "structural")`.

In addition to value templates and variables, terms can also be defined recursively, by using a term operator (`ONEOF`, `LISTOF`, `SETOF`, `MAX`, `MIN`), or for some special cases can even be assigned a null value (with the help of the keyword `NUL`).

6.2.5 Templates

Templates represent the most essential part of a mapping. A template provides the data source from which the value of a target attribute will be constructed. This can be as simple as an ordinary source attribute referenced by name, and as complex as a series of nested *template operators*.

The general syntax for templates is as follows:

```
valueTemplate    = simpleTemplate | literal |
                   template_op |
                   lisp_expr | searchExpression .
simpleTemplate   = attr | CLASSNAME | OBJECTNAME |
                   [ class addr_op ] THIS .
```

According to this definition, a *value template* can be a simple template, a literal constant of type LOGICAL, INTEGER, REAL or STRING, a self-contained common LISP expression identified by the reserved word `LISP`, i.e. `(LISP` *lisp-form*`)`, a knowledge-based search expression as described in section 4.7, or a template operator.

The common case of use of simple templates is the direct referencing of attributes by name. Besides that, a simple template may contain also a reference to the currently processed item (given by the keyword `THIS`), the name of the current class (specified by the reserved word `CLASSNAME`) or the name of the current object instance (`OBJECTNAME`) represented as strings. The last two cases are needed when the name of an object has to be stored in the attribute of another object. This can be observed in many simpler data models where such “string referencing” is implemented.

The most interesting part of the template specification are the *template operators*.

In general, a template operator always represents some kind of a **mapping pattern**. When applied, it not only returns the value(s) specific for that pattern, but may also play a role in determining the instance mapping set for a whole class, as well as the cardinality of the class mapping. This is so, because in most cases the cardinality of a mapping cannot be defined explicitly on class level but results, implicitly, from the specific types of the attribute equivalences. With the help of template operators, such “side effects” are made more transparent both to the end-user and to the mapping engine.

The available **template operators** in CSML have been developed in the course of several iterations on the basis of the analysed mapping patterns and experience. Currently, the following nine operators are defined:

```
template_op    = apply_op | mapcar_op |
                 user_input_op | user_choice_op |
                 descendants_op | ref_op |
                 assoc_op | new_op | iter_op .
```

The `apply_op` and the `mapcar_op` support the category of functional mapping patterns, whereas `user_input_op` and `user_choice_op` support the generative user-dependent mapping patterns. The remaining five template operators are designed especially for the complex patterns defined in table 5.6.

In the following, each of these operators will be discussed briefly, and as simple as possible examples will be given to demonstrate its intended use. To keep the description concise, the referenced attributes in these examples are assumed to be intuitively clear from the context. More detailed examples are given later in this chapter, and detailed case studies involving all of the listed operators are presented in the next chapter 7.

The **`apply_op`** simply enables the use of a function where a term is expected. Its syntax is:

```
apply_op       = '(' APPLY fn_ref ')' .
```

For example, given that a point has to be transformed from cartesian to polar coordinates, the following construction can be used to obtain the value of the radius:

```
(MAP CLASS PolarPoint FROM CartesianPoint
 ATTRIBUTES
   (MAKE radius FROM (APPLY (LAMBDA (x y)
                              (SQRT (+ (* x x) (* y y))))
                             ARGS X_coord Y_coord)) … )
```

The **mapcar_op** resembles the MAPCAR function of Common LISP. It provides a type of iteration in which the function given by fn_ref is successively applied to one or more aggregations (lists or sets). The result of that iteration is a list containing the respective results of the function application on the elements of the aggregations supplied as arguments to MAPCAR. The function must take the same number of arguments as the arguments of MAPCAR, which must be lists or sets of the same length; if not, the iteration terminates when the shortest list or set runs out, and the excessive elements are simply ignored.

```
mapcar_op       = '(' MAPCAR fn_ref ')' .
```

For example, if the reactions of columns are specified as lists of reals (in pounds) in one model and have to be applied to foundation elements as actions (specified in kN) in another model, the following construction can be used:

```
(MAP CLASS FoundationElement FROM Column
 ATTRIBUTES
  (MAKE load_actions FROM (MAPCAR (LAMBDA (x)
                                   (* 0.00454 (- x)))
                                  ARGS reactions_list)) … )
```

The **user_input_op** provides a possibility to supply values interactively during a mapping. In this way, it is possible to fulfil a mapping with the help of the end-user that cannot otherwise be defined completely because of only a few missing data in the source model. It is on the responsibility of the application which issues the mapping request to construct the respective user dialog. The default implementation in the mapping engine assumes only a simple generic function that can be used in LISP-based applications.

The syntax of user_input_op is quite simple. It provides the minimum of data that would allow a simple realisation of the template. These are: the data type of the expected input value(s), an optional prompt message, a term which will be assigned the provided value and an optional default value for the case that the user does not provide any input. The possibility to use a more advanced dialog feature is given by the RUN clause which enables to call an appropriate application-specific function.

```
user_input_op   = '(' USER-INPUT input_type
                      { prompt_string | RUN fn_file_name }
                      { term }*
                      [ DEFAULT { term }+ ] ')' .
```

For example, to specify an overall foundation depth for a foundation design model when such data are missing in the source model, the following can be written:

```
(MAP CLASS BuildingFoundation FROM Building
 ATTRIBUTES
  (MAKE foundation_depth FROM
    (USER-INPUT REAL "Found. depth <0.0>:" DEFAULT 0.0)) … )
```

The **user_choice_op** is very similar to the user_input_op. It provides the possibility to select an input value from a list of possible values. This can be a list of constants, but also any possible list of values contained in the source model. Another typical case is to use as a “select list” an enumeration type specified in the target model.

user_choice_op returns either one value (when the keyword ONEOF is specified), or multiple values collected in a list (when the keyword LISTOF is given).

```
user_choice_op  = '(' USER-CHOICE { ONEOF | LISTOF } { term }+
                      { prompt_string | RUN fn_file_name }
                      { term }*
                      [ DEFAULT { term }+ ] ')' .
```

Continuing the example from above, in order to specify a preferred type for the foundation system the following can be written[*]:

```
(MAP CLASS BuildingFoundation FROM Building
 ATTRIBUTES
  (MAKE foundation_type FROM
    (USER-CHOICE ONEOF (LISTOF SPREAD_FOOTINGS
                               STRIP_FOOTINGS
                               PILES MAT UNDEFINED))) … )
```

If in the target model the list of permitted foundation types is defined as an enumeration with the name `foundation_types`

```
    (USER-CHOICE ONEOF foundation_types)
```

could be used instead.

The **descendants_op** is now the first (and most simple) of the template operators supporting the complex mapping patterns given in table 5.6. It provides a local form of grouping, similar to the `GROUPS` clause of a class mapping declaration, but does not require the use of an additional variable.

```
descendants_op  = '(' DESCENDANTS class
                        [ FOR { simpleTemplate }+ ]
                        [ local_cond_spec ] ')' .
```

For example, to group all continuous beams in a building structure and use them as a selection list in a `user_choice_op` the following can be written:

```
…
(USER-CHOICE ONEOF (DESCENDANTS beam
                    WHEN (beam_type = "continuous beam"))) …
```

The application of the **ref_op** is also fairly obvious. It allows to use attributes of an object which is referenced through a pointer contained in an attribute of the current object. By embedding `REF` templates in one another, any level of referencing can be specified.

```
ref_op          = '(' REF attr FOR { term }+ ')' .
```

For example, to store directly the name of a building material in the attribute `material_name` of a `BuildingElement` instance in the target model when similar instances in the source model are linked to respective “material” objects with more detailed properties, the following construction can be applied:

```
(MAP CLASS BuildingElement FROM BuildingElement
 ATTRIBUTES
  (MAKE material_name FROM (REF material FOR name)) … )
```

The rationale of the **assoc_op** is more complicated. Its goal is to enable the specification of a whole range of patterns where *inter-related mappings* are involved. Such situations

[*] In these examples for `USER-CHOICE` the required prompt string is omitted for brevity.

occur when an attribute of the current object references an instance of another object, and the reference structure has to be reflected in the target model as well.

In the simplest case of 1:1 correspondence, i.e. when both the source and the target objects reference (through their respective attributes) single other objects, the `assoc_op` is not needed as this mapping can be accomplished simply by an `identity_decl`, i.e. (SAME a_T AS a_S), which works also on pointer types. However, when one or both of the considered attributes are lists, the situation can be much more complicated, with a lot of different variations involved. In such cases, an `assoc_op` must be applied.

In general, `ASSOC` requires first the class that is being referenced and then the name of the attribute by which this reference is done (`REF` clause). In addition, it allows to define also local conditions to constrain the association between the entities, as well as to use the reference attribute itself as a reference to another attribute (`FOR` clause). The latter is actually a rarely needed construct which is provided only for convenience, as a shortcut to an embedded `REF` template that would otherwise be needed.

assoc_op = '(' ASSOC { class }$^+$ [REF attr]
[FOR { simpleTemplate }$^+$] [local_cond_spec] ')'.

As an example, consider a situation where for a composite building element, e.g. a wall, a number of building materials are defined, and the pointers to the respective “material” instances are stored as a list in the attribute `ConsistsOf`. In the target model, the same structure has to be reflected in the attribute `StructMaterials`, but only for component materials that are instances of the classes `CONCRETE` or `STEEL`. Obviously, this will lead to a M:N correspondence of the “material” objects with respect to the attributes `ConsistsOf` and `StructMaterials`. To achieve this, the following construction can be applied:

```
(COPY CLASS Material)
 …
(MAP CLASS StructuralElement FROM BuildingElement
 ATTRIBUTES
  (MAKE StructMaterials FROM
    (ASSOC Material REF ConsistsOf
                   WHEN ONEOF (CLASSNAME = "CONCRETE")
                              (CLASSNAME = "STEEL"))) … )
```

The **new_op** is complementary to `ASSOC`. It has a similar syntax, but is used in cases where a reference has to be established in the target model which does not exist in the source model. In order to construct such references, the addressed target objects must first be created and both the source and the target instance sets must then be analysed. Therefore, unlike `ASSOC`, the evaluation of a `NEW` template must always be deferred until all “directly resolvable” mappings are performed. The referencing of such “new” target objects may often be accompanied by one or more fairly complex local conditions.

new_op = '(' NEW class [REF attr]
[FOR { simpleTemplate }$^+$] [local_cond_spec] ')'.

A simple example from IFC is to populate the attribute `RelatedBuildings` of the entity `IfcRelServicesBuildings` in the case when a functional system, e.g. an HVAC piping system, “services” all buildings in a model. This can be expressed as:

```
 …
(MAKE RelatedBuildings FROM (NEW IfcBuilding)) …
```

At last, the **iter_op** enables the specification of an *iterator* which scans the values stored in an attribute of list or set type in a source instance and populates with the individual values found distinct simple type attributes in respectively created distinct target instances. Thus, unlike the mapcar_op which also iterates over lists and sets, the iter_op establishes a 1:N correspondence between the source and the target instances in the given mapping, whereas MAPCAR merely creates one corresponding list (or set) in the respective attribute of the target.

If more than one iter_op is used in one mapping declaration, the result will be a cross product of the separate 1:N mappings. In such case, the pruning of identical instances must be performed by the mapping engine as a last step in the mapping task. CSML does not provide an explicit construct to specify when and how the result set should be pruned.

```
iter_op          = '(' ITERATE-ON { term }+
                         [ local_cond_spec ]')' .
```

For example, if in a source model all used materials are collected in one composite object, ProductMaterials, which contains a list of their standard designated names in the attribute MaterialNames, but in the target model each such material should be stored in a separate instance Material with its name stored in an attribute called Name, the following construction can be used to achieve the needed transformation:

```
(MAP CLASS Material FROM ProductMaterials
 ATTRIBUTES
  (MAKE Name FROM (ITERATE-ON MaterialNames)) … )
```

6.2.6 Functions

The use of in-line LAMBDA functions has already been addressed at several places in this section. It remains to show how other types of functions loaded from external source or executable files can be used.

The general syntax of a function reference is as follows:

```
fn_ref           = fn_descr [ ARGS { term }+ ] .
fn_descr         = fn_name | lisp_lambda_fn .
fn_name          = symbol .
```

As shown, a LISP LAMBDA function is just one of the possibilities to use procedural code in CSML. The more general way is to pre-load all needed functions using the PRESETS section of a schema mapping declaration and then merely reference them by name. For example:

```
(MAKE line_length CONSTRUCTOR vector_length ARGS x y z)
```

The syntax for loading such external functions is:

```
ext_fn_load_op   = '(' LOAD fn_file_name
                         [ FOR { fn_name }+ ] ')' .
```

The arguments consumed by a function must be supplied in the ARGS clause of fn_ref. Such arguments can be any allowable data objects. The responsibility that they correspond to the formal parameters of the function is on the developer of the mapping specification.

In the current version of CSML, external functions can be defined only in LISP and, with some restrictions, in C. The use of other programming languages (an implementational, and not a principal issue) has not been in the scope of this research.

6.2.7 Expressions

Except for a few directly defined relations between terms, CSML does *not* support expressions in the sense of a normal programming language. There are two reasons for that. The first, more pragmatic reason is that the definition of expressions would overburden the syntax and make the mapping parser work slower. The second, more important reason is that CSML is intentionally designed with focus on the principal relationships between conceptual models, and not on eventual procedural operations that might be needed in certain attribute transformations. In my opinion, a mapping specification ruffled up with arithmetic expressions can only impair the overall readability and blur the objectives of the mapping task. The use of external functions for all algorithmic computations corresponds better to the purposes of a mapping language.

6.2.8 Mapping domains and mapping extents

Whilst all described mapping constructs are defined generically, on schema level, the actual mapping transformations are not performed on classes and their attributes, but on entity instances and the data they contain. Each such mapping transformation is associated with a mapping domain and a mapping extent.

A *mapping domain* contains all source instances to which the given mapping equivalences can be applied, i.e. if S is a source class and T is the respective target class, such that $S \to T$, then $\mathbf{D}_{map} = \{ s_i \} \subseteq S \Leftrightarrow \forall s_i \in S \; \exists t_j \in T : s_i \to t_j$.

A *mapping extent* is comprised of all possible combinations of source-target instances fulfilling the given mapping equivalences, i.e.: $\mathbf{E}_{map} = \bigcup_{i,j} s_i \to t_j \; (i \times j)$.

The size of the mapping domain is affected by the number of source classes participating in the mapping, by the given conditions and by the specified groupings. If there are no conditions and groupings in a class mapping declaration, the mapping domain will have a size equal to the number of instances of the source class (when only one source class is involved), or to the cartesian product of the instances of all source classes (when multiple source classes are specified in the FROM clause of the declaration). Conditions and groupings suppress the creation of a full cartesian product and consequently reduce the size of the mapping domain.

The size of the mapping extent depends on the mapping domain and on the multiplicity of the specified attribute mappings. However, different from the relationships between entity domains in relational databases, in a mapping initially only the instances of the source classes exist, whereas the instances of the target class T have to be deduced from the applied mapping equivalences.

In particular, the size of the mapping extent can be influenced by the template operators ASSOC (depending on the multiplicity of the specific associations), NEW (depending on other class mappings involving the referenced “new” target instances) and ITERATE-ON (which results always in a 1:N correspondence). In each of these cases, the size of the extent will be a multiple of the combined effect of such operations. As the target instances do not exist at the beginning and can thus be determined only on the basis of the mapping declarations, it is possible that after the mapping the extent contains duplicate instances of the target class. Therefore, in a last step, the resulting extent has to be analysed and all duplicates have to be removed. This task is on the responsibility of the implemented mapping engine.

6.3 Formal Syntax Specification

After the given outline of the structure and the basic components of CSML, in this section the full technical specification of its syntax is presented in three parts: (1) *tokens*, providing low-level definitions for all allowed lexical elements, (2) *grammar rules* providing all high-level structures, and (3) *informal propositions*, including additional information not covered in the formal specifications for conciseness, or because it represents imported constructs described in other sections of this thesis or in other literature sources.

The primary rule of the CSML syntax is rule (80).

Tokens

The following productions define the tokens of the language used in all subsequent grammar rules. Unless explicitly stated, no `whitespace` characters are allowed within the text matched by a single syntax rule for a token.

Character classes:

The specification of character classes is the same as for the Information Container externalisation and the knowledge-based expressions given in sections 4.2 and 4.7 respectively. It is repeated here for convenience.

```
(1)  whitespace   = ASCII-SP | ASCII-HT | ASCII-CR | ASCII-LF |
                    ASCII-FF .
(2)  letter       = 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' |
                    'h'.| 'i' | 'j' | 'k' | 'l' | 'm' | 'n' |
                    'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' |
                    'v' | 'w' | 'x' | 'y' | 'z' |
                    'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' |
                    'H'.| 'I' | 'J' | 'K' | 'L' | 'M' | 'N' |
                    'O' | 'P' | 'Q' | 'R' | 'S' | 'T' | 'U' |
                    'V' | 'W' | 'X' | 'Y' | 'Z' .
(3)  digit        = '0' | '1' | '2' | '3' | '4' | '5' | '6' |
                    '7' | '8' | '9' .
(4)  special_char = '!' | '″' | '$' | '%' | '&' | '.' | ',' |
                    '*' | '+' | '-' | '(' | ')' | '[' | ']' |
                    '{' | '}' | '<' | '>' | '=' | '@' | '?' |
                    '#' | '^' | ':' | ';' | ''' | '`' | '~' |
                    '|' | '/' | '_' .
(5)  escape_char  = '\' .
(6)  escape_seq   = '\n' | '\r' | '\t' | '\\' .
(7)  quote_char   = '\″' .
(8)  non_q_char   = letter | digit | special_char | escape_seq .
```

Keywords:

The below list presents all keywords of CSML. Although, technically speaking, several of these keywords may appear only in a specific context and may thus be 're-used' e.g. as class, instance or attribute names at other places, it is safer and simpler to consider all defined keywords as **reserved words** in CSML. Rule (74) provides an alternative syntax for the cases where a symbol defined in a model schema is identical with a keyword.

```
(9)  AFTER       = 'AFTER'.
(10) ALL         = 'ALL'.
(11) ALLBUT      = 'ALLBUT'.
(12) APPLY       = 'APPLY'.
(13) ARGS        = 'ARGS'.
(14) AS          = 'AS'.
(15) ASSOC       = 'ASSOC'.
(16) ATTRIBUTES  = 'ATTRIBUTES'.
(17) BEFORE      = 'BEFORE'.
(18) BOOLEAN     = 'BOOLEAN'.
(19) CLASS       = 'CLASS'.
(20) CLASSES     = 'CLASSES'.
(21) CLASSNAME   = 'CLASSNAME'.
(22) COMMENTS    = 'COMMENTS'.
(23) CONDITIONS  = 'CONDITIONS'.
(24) CONSTRUCTOR = 'CONSTRUCTOR'.
(25) COPY        = 'COPY'.
(26) DEFAULT     = 'DEFAULT'.
(27) DEPENDENTS  = 'DEPENDENTS' |
            'DEPENDENT-CLASSES'.
(28) DESCENDANTS = 'DESCENDANTS'.
(29) EXCLUSIONS  = 'EXCLUSIONS'.
(30) FALSE       = 'false' |
                   'FALSE'.
(31) FOR         = 'FOR'.
(32) FROM        = 'FROM'.
(33) GROUPS      = 'GROUPS'.
(34) INTEGER     = 'INTEGER'.
(35) ITERATE-ON  = 'ITERATE-ON'.
(36) LISP        = 'LISP'.
(37) LISTOF      = 'LISTOF'.

(38) LOAD        = 'LOAD'.
(39) LOGICAL     = 'LOGICAL'.
(40) MAKE        = 'MAKE'.
(41) MAP         = 'MAP'.
(42) MAPCAR      = 'MAPCAR'.
(43) MAX         = 'MAX'.
(44) MIN         = 'MIN'.
(45) NEW         = 'NEW'.
(46) NUL         = 'NUL'.
(47) NUMBER      = 'NUMBER'.
(48) OBJECTNAME  = 'OBJECTNAME'.
(49) ONEOF       = 'ONEOF'.
(50) PARTIALLY   = 'PARTIALLY'.
(51) PRED        = 'PRED'.
(52) PRESETS     = 'PRESETS'.
(53) REAL        = 'REAL'.
(54) REF         = 'REF'.
(55) RUN         = 'RUN'.
(56) SAME        = 'SAME'.
(57) SETOF       = 'SETOF'.
(58) SHARED      = 'SHARED'.
(59) STRING      = 'STRING'.
(60) THIS        = 'THIS'.
(61) TRUE        = 'true' |
                   'TRUE'.
(62) UNKNOWN     = 'unknown' |
                   'UNKNOWN'.
(63) USER-CHOICE = 'USER-CHOICE'.
(64) USER-INPUT  = 'USER-INPUT'.
(65) VAR         = 'VAR'.
(66) WHEN        = 'WHEN'.
```

Lexical elements:

Most of the lexical elements in CSML are also very similar or even the same as in the Information Container specification presented in section 4.2. However, analogous to the specification of knowledge-based expressions, CSML provides also the possibility to use *free variables* (rule 75) and *operators* (rules 76-78). Free variables are bound at execution time to respectively computed values in a similar way as in knowledge-based expressions. Together with operators and functions they allow to express complex correspondences between the elements of the source and target schemas, as well as to take in consideration certain context-dependent data transformations.

```
(67) boolean    = TRUE | FALSE .
(68) logical    = TRUE | FALSE | UNKNOWN .
(69) longint    = [ sign ] { digit }+ .
(70) real       = [ sign ] { digit }+ '.' { digit }*
                  [ { 'E' | 'e' } [ sign ] { digit }+ ] .
(71) string     = '"' { non_q_char | quote_char | ' ' }* '"' .
(72) simpleType = BOOLEAN | INTEGER | LOGICAL | NUMBER | REAL | STRING .
```

```
(73) symbol    = letter { letter | digit | '-' | '_' }* .
(74) keyword   = [ '_' ] symbol .
(75) variable  = '?' symbol .
(76) term_op   = ONEOF | LISTOF | SETOF | MAX | MIN .
(77) rel_op    = '=' | '<' | '>' | '<=' | '>=' | '<>' | ':=:' .
(78) addr_op   = '->' .
(79) sign      = '+' | '-' .
```

Grammar rules

The high-level production rules given below specify how the tokens of CSML can be combined into valid statements in a mapping specification. In order to avoid ambiguity, `whitespace` characters *should* be used as separators between the individual tokens.

```
(80)  schema_mapping_decl     = '(' MAP [ PARTIALLY ] { schema }+
                                      FROM { schema }+
                                      mapping_spec_set ')' .
(81)  mapping_spec_set        = { { class_mapping_spec }+ |
                                    dependent_classes_spec |
                                    presets_spec | comments_spec }+ .
(82)  class_mapping_spec      = CLASSES { class_mapping_decl }+ .
(83)  dependent_classes_spec  = DEPENDENTS { class_mapping_decl }+ .
(84)  presets_spec            = PRESETS { presets_decl }+ .
(85)  comments_spec           = COMMENTS { string }+ .
(86)  class_mapping_decl      = copy_class_decl | map_class_decl .
(87)  copy_class_decl = '(' COPY [ CLASS ] class [ FROM class ]
                            { cond_spec | exclusion_spec }* ')' .
(88)  map_class_decl  = '(' MAP [ CLASS ] class FROM { class }+
                            { { attr_map_spec }+ |
                              group_spec |
                              var_spec |
                              cond_spec |
                              exclusion_spec }+ ')' .
(89)  presets_decl    = ext_fn_load_op | ext_fn_run_op |
                        variable_decl .
(90)  ext_fn_load_op  = '(' LOAD fn_file_name
                            [ FOR { fn_name }+ ] ')' .
(91)  ext_fn_run_op   = '(' RUN [ BEFORE | AFTER ] fn_file_name
                            [ FOR { fn_name }+ ] ')' .
(92)  group_spec      = GROUPS { group_decl }+ .
(93)  group_decl      = '(' { class }+ FOR variable
                            [ local_cond_spec ] ')' .
(94)  var_spec        = VAR { variable_decl }+ .
```

```
(95)  variable_decl   = '(' MAKE { variable }+
                            { FROM { term }+ [ suffix_op ] |
                              CONSTRUCTOR fn_ref } ')' .
(96)  cond_spec       = CONDITIONS [ ALL | ONEOF ] { cond_decl }+ .
(97)  local_cond_spec = WHEN [ ALL | ONEOF ] { cond_decl }+ .
(98)  cond_decl       = '(' term rel_op term ')' |
                        '(' PRED { fn_ref | KB_Template } ')' .
(99)  exclusion_spec  = EXCLUSIONS { exclusion_decl }+ .
(100) exclusion_decl  = '(' class [ FOR { attr }+ ] ')' .
(101) attr_map_spec   = ATTRIBUTES { attr_map_decl }+ .
(102) attr_map_decl   = identity_decl | equivalence_decl .
(103) identity_decl   = '(' SAME { { attr }+ [ AS { attr }+ ] |
                                   ALL | ALLBUT { attr }+ } ')' .
(104) equivalence_decl = '(' MAKE { attr }+
                            { FROM { term }+ [ suffix_op ] |
                              CONSTRUCTOR fn_ref }
                            [ DEFAULT { term }+ ] ')' .
(105) term            = valueTemplate | variable |
                       '(' term_op { term }+ ')' | NUL .
(106) valueTemplate   = simpleTemplate | literal | template_op |
                        lisp_expr | searchExpression .
(107) simpleTemplate  = attr | CLASSNAME | OBJECTNAME |
                        [ class addr_op ] THIS .
(108) literal         = boolean | logical | longint | real | string |
                        symbol .
(109) template_op     = apply_op | assoc_op | descendants_op |
                        iter_op | mapcar_op | new_op | ref_op |
                        user_choice_op | user_input_op .
(110) apply_op        = '(' APPLY fn_ref ')' .
(111) assoc_op        = '(' ASSOC { class }+ [ REF attr ]
                            [ FOR { simpleTemplate }+ ]
                            [ local_cond_spec ] ')' .
(112) descendants_op  = '(' DESCENDANTS class
                            [ FOR { simpleTemplate }+ ]
                            [ local_cond_spec ] ')' .
(113) iter_op         = '(' ITERATE-ON { term }+
                            [ local_cond_spec ] ')' .
(114) mapcar_op       = '(' MAPCAR fn_ref ')' .
(115) new_op          = '(' NEW class [ REF attr ]
                            [ FOR { simpleTemplate }+ ]
                            [ local_cond_spec ] ')' .
(116) ref_op          = '(' REF attr FOR { term }+ ')' .
(117) user_choice_op  = '(' USER-CHOICE { ONEOF | LISTOF } { term }+
                            { prompt_string | RUN fn_file_name }
                            { term }*
                            [ DEFAULT { term }+ ] ')' .
```

```
(118) user_input_op  = '(' USER-INPUT input_type
                           { prompt_string | RUN fn_file_name }
                           { term }*
                           [ DEFAULT { term }+ ] ')' .
(119) suffix_op      = APPLY fn_descr | MAPCAR fn_descr |
                       MAP result_type fn_descr .
(120) lisp_expr      = '(' LISP lisp_form ')' .
(121) schema         = [ SHARED ] schema_name .
(122) class          = [ schema_name ':' ] symbol | variable .
(123) attr           = [ class addr_op ] symbol | variable .
(124) fn_ref         = fn_descr [ ARGS { term }+ ] .
(125) fn_descr       = fn_name | lisp_lambda_fn .
(126) schema_name    = symbol .
(127) fn_name        = symbol .
(128) fn_file_name   = string .
(129) result_type    = symbol | simpleType .
(130) input_type     = simpleType .
(131) prompt_string  = string .
```

Informal propositions

The propositions listed below define additional rules for the proper use of CSML that cannot be readily expressed in EBNF.

1) **Entities defined in EXPRESS data models** subject to an inter-model mapping specification are referenced in CSML as follows:
   - Entity classes are represented as symbols, eventually prefixed with the respective schema name to avoid possible ambiguity when the same class name is found both in a source and a target model (see rule 122);
   - Attribute names are also represented as symbols and may be prefixed with the class and schema name for similar reasons (see rule 123);
   - References (pointers) to specific entity instances and/or attribute values need not be represented in CSML because all mapping specifications are provided on class level; if necessary, free variables can be used for intermediate storage of such values at execution time.
2) **`longint`** values may contain as many digits as permitted in the internal representation of long integer numbers, normally in the range $[-2^{64} ; 2^{64} -1]$.
3) **`real`** values are interpreted internally as double precision floating point numbers.
4) The keyword **THIS** represents a pointer to the currently processed object in the context in which it appears, and the keywords **OBJECTNAME** and **CLASSNAME** represent respectively the symbolic name of this object or of its class as strings.
5) Although CSML does not enforce any requirement w.r.t. the **internal format of an object name**, in the context of the proposed environment an object name will be represented always by the object’s refID as defined in the Information Container specification, i.e.

```
refID   = modelID | objID .
modelID = ID .
objID   = [ modelID ':' ] ID .
ID      = symbol [ '.' { symbol | longint } [ version ] ] .
```

6) **`fn_ref`** and **`fn_descr`** defined in rules (124-125) provide the possibility to include 'foreign' code in a mapping specification as a substitution for any more sophisticated rules that might be needed for the definition of arithmetic or relational expressions. Such foreign code may be specified in an *external function file* or, in more simple cases, as an *inline LISP lambda function*[*)]. The use of `fn_ref` (in rules 95, 98, 104, 110, 114) is different from the use of `fn_descr` (in rule 119) w.r.t. the treatment of the function arguments. `fn_ref` assumes that an explicit argument list is provided, whereas the arguments of a call to a function given by its function descriptor (`fn_descr`) are constructed automatically from the associated list of terms. In both cases, there are two “automatic arguments” that are always passed to the function: these are the source and the target object classes given in the class mapping declaration[**)].

   Each foreign function referenced in such way should always return a value of the appropriate data type required at the place where it is used. In particular, when a function is used as a predicate in a condition declaration (rule 98), it is expected to return only one single boolean value.

7) ***lisp_form*** and ***lisp_lambda_fn***, referenced in rules (120) and (125) respectively, provide a second alternative for the use of external constructs in CSML.

   A *lisp_form* can be any valid Common LISP top-level form as defined in (Steele 1990), but it should always return value(s) of the appropriate data type required at the place where it is used. It may include:

   – local variable bindings introduced by the Common LISP ***let*** construct;
   – free variables as defined in rule (75) above;
   – any standard Common LISP function, not referencing a user-defined routine;
   – object/attribute accessor functions as implemented in the mapping engine.

   A *lisp_lambda_fn* can be used at all places where a `fn_descr` is expected. However, different from a foreign function defined in an external function file, a *lisp_lambda_fn* is passed only the arguments that are explicitly given in its definition, i.e. it does not consume the automatic arguments provided for the source/target classes in an inter-class mapping declaration.

8) ***KB_Template*** and ***searchExpression***, referenced in rules (98) and (106) respectively, offer yet another possibility for specifying more sophisticated expressions in CSML. They provide the necessary link allowing to use the knowledge-based extensions of the project data server within a mapping, following the syntax given in section 4.7.

   A *searchExpression* may be specified at any place where a `valueTemplate` is expected, its use in a mapping declaration is similar to that of a LISP expression. In contrast, a *KB_Template* may be used only as a predicate function in a condition statement and should always return *one* boolean value, i.e. it must express a fact to be checked against the actual state of the project data. No free variables are allowed in this specific use of a *KB_Template*.

[*)] In the currently prototyped Mapping Engine based on CSML only Common LISP and C functions may be defined as foreign functions in a mapping. The use of another programming language, though principally possible, requires a lot more implementation efforts.

[**)] When there are more than one source classes involved, they are first packed as a list and then passed to the function as one single argument.

9) Certain grammar rules of CSML imply the use of **lists**. Whilst the syntax of such rules does not impose any limitations on the allowable length of such lists by itself, there are several dependencies that have to be observed in the actual use of lists in a mapping definition:

   – The list of terms in the FROM clause of `variable_decl` (rule 95) should match the list of variables specified in its MAKE clause;
   – The list of attributes in the AS clause of an `identity_decl` (rule 103) should correspond to the list of attributes to be mapped; the same applies also to the lists of terms given in the FROM clause and the DEFAULT clause of an `equivalence_decl` (rule 104);
   – The list of attributes in the FOR clause of an `assoc_op` (rule 111) should correspond to the specified list of classes, and each referenced attribute must be an existing attribute of the respective class in the class list;
   – Each term specified in the optional DEFAULT clause of a `user_choice_op` (rule 117) should be identical with one of the terms in the 'choice' list, and all terms in the 'choice' list must be different from each other;
   – The list of arguments in the ARGS clause of a `fn_ref` (rule 124) should correspond in number and type to the actually defined arguments of the function, with an offset of two for the automatically supplied arguments for the source and target classes.

10) Finally, although not required by CSML, **upper and lower case** in all names defined in EXPRESS schemas should be observed for compatibility with applications relying on case-sensitive names, such as C++ or Java-based programs.

The above set of informal propositions, complementing the formal grammar rules from the preceding section, complete the specification of CSML. With the help of the developed formalism (and with the help of a *Mapping Parser* implemented as part of the prototyped environment) each mapping model specified in CSML can be readily analysed, converted to an appropriate computer-interpretable format and supplied as input to the mapping engine.

It remains to be proven that the developed formalism is sufficient to cover the mapping problems addressed in chapter 5.

In particular, it is necessary to show that with the help of CSML it is possible:

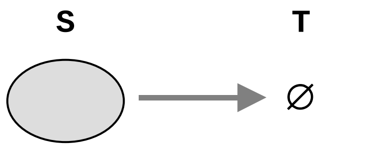
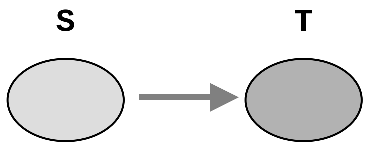
– to describe correctly all mapping patterns identified in section 5.6,
– to express properly all relevant relational algebra operations known from database research,
– to solve adequately typical mapping problems detailed in other research studies conducted on the mapping arena, and
– to provide appropriate coverage of practical tasks from the AEC domain.

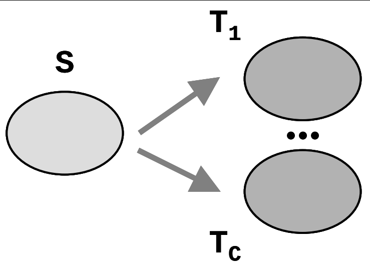
The first three of the above issues are addressed in the following three sections of this chapter. The last issue is discussed in the next chapter, on the basis of selected case studies.

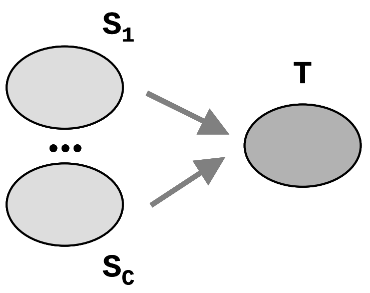
6.4 Representation of Basic Mapping Patterns

In order to prove that CSML is capable to represent the mapping patterns introduced in chapter 5, tables 5.3 to 5.7 are re-used here in a slightly different form. The last column in the original tables, describing the meaning of each of the discussed patterns, is substituted with the CSML specification which can be used to express the same mapping patterns *formally*. To focus the attention, the qualifying constructs are underlined. The first two columns are reproduced from chapter 5 to avoid cross-referencing.

Table 6.1: Representation of the unconditional class level mapping patterns in CSML

Class mapping pattern	Schematic presentation of the mapping transformations	CSML specification
1 : ∅		No operation needed
1 : 1		`(COPY CLASS T FROM S)`

1 : C		`(MAP CLASS` T_1 `FROM S ... )`
		`...`
		`(MAP CLASS` T_C `FROM S ... )`
		with concrete attribute mappings detailed in the `ATTRIBUTES` sections

C : 1		`(MAP CLASS T FROM` S_1 `...` S_C `... )` with concrete attribute mappings detailed in the `ATTRIBUTES` sections

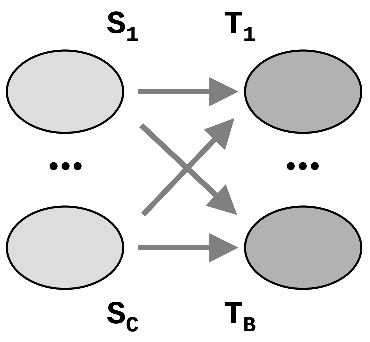
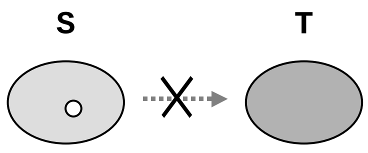
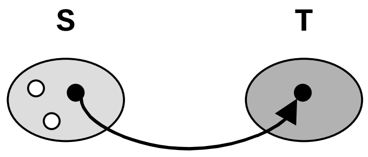
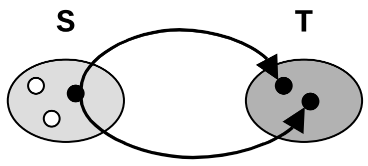
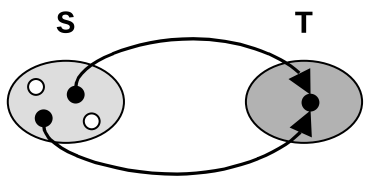
C : B		`(MAP CLASS` T_1 `FROM` S_1 `...` S_C `... )`
		`...`
		`(MAP CLASS` T_B `FROM` S_1 `...` S_C `... )`
		with concrete attribute mappings detailed in the `ATTRIBUTES` sections

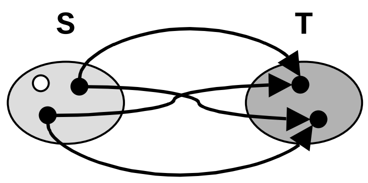
Table 6.2: Representation of the conditional instance level mapping patterns in CSML

Instance mapping pattern	Schematic presentation of the mapping transformations	CSML specification
1 : ∅		No operation needed

1 : 1		**(COPY CLASS T FROM S**
		  CONDITIONS ***$cond_1$*** **. . .** ***$cond_N$***
		  ...
		)

1 : C
1 : N
		**(MAP CLASS T FROM S**
		  CONDITIONS ***$cond_1$*** **. . .** ***$cond_N$***
		  ATTRIBUTES
		    (MAKE a_t FROM (ITERATE-ON a_S))
		     ...
		)

C : 1
N : 1
		**(MAP CLASS T FROM S**
		  GROUPS (S FOR ?S
		          WHEN
		          ONEOF ***$cond_1$*** **. . .** ***$cond_N$*** **)**
		  ATTRIBUTES
		  ...
		)
		Note: The variable ?S must be used at least once in the ATTRIBUTES section to achieve the needed effect (see e.g. table 6.4, first pattern)

C : B
C : N
N : C
N : M
		**(MAP CLASS T FROM S**
		  GROUPS (S FOR ?S
		          WHEN
		          ONEOF ***$cond_1$*** **. . .** ***$cond_N$*** **)**
		  ATTRIBUTES
		    (MAKE a_T FROM
		        (ITERATE-ON ?S -> a_S))
		    ...
		)
		Note: In this case, which is effectively a combination of the previous two, ?S defines on the one hand side the grouping of the instances of S, and on the other hand side provides an iterator creating several instances of T depending on the value of a_S. It is not possible to use more than one grouping in such way because the result may be an undefinable set of instances to map.

The next three tables depict how the various **attribute mapping patterns** can be represented with the help of CSML. For conciseness, only the attribute mappings $a_S \rightarrow a_T$ included in the `ATTRIBUTES` section of a class mapping declaration are shown in the column presenting the CSML specification. **D**a_S and **D**a_T represent the definitional domains for a_S and a_T respectively.

Table 6.3: Representation of the basic attribute level mapping patterns in CSML

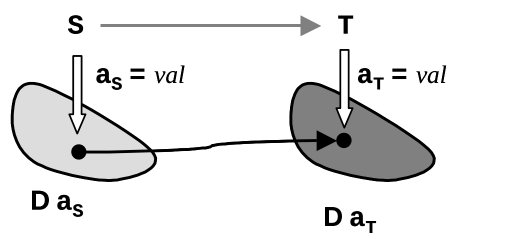
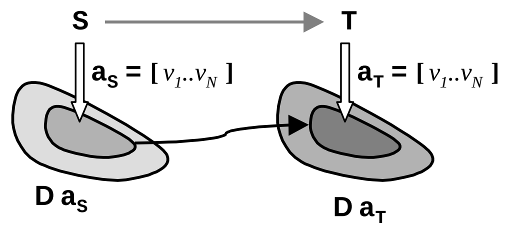
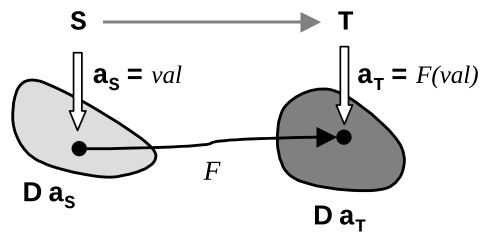
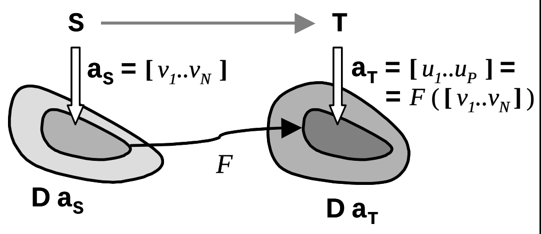
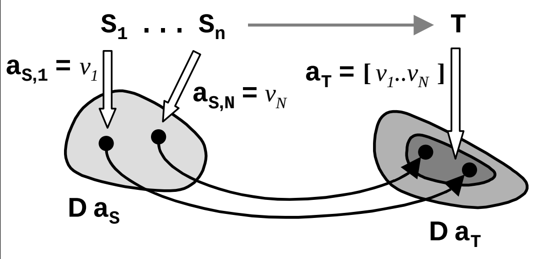
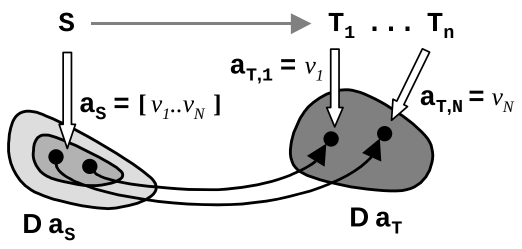
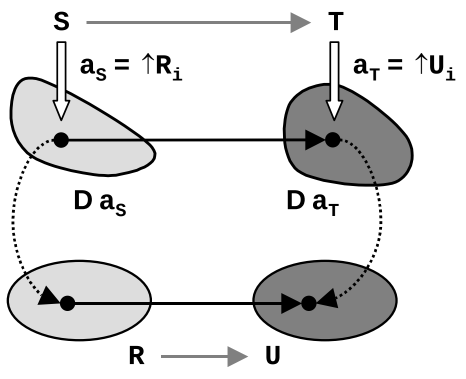
Attribute mapping pattern	Schematic presentation of the mapping transformations	CSML specification
Identity (simple equivalence)	 S T a_S = *val* a_T = *val* D a_S D a_T	(**SAME** a_T **AS** a_S)
Aggregate identity (simple set equivalence)	 S T $a_S = [v_1..v_N]$ $a_T = [v_1..v_N]$ D a_S D a_T	(**SAME** a_T **AS** a_S)
Functional equivalence	 S T a_S = *val* a_T = *F(val)* *F* D a_S D a_T	(**MAKE** a_T **FROM** (**APPLY F ARGS** a_S)) or, as a defered suffix operation: (**MAKE** a_T **FROM** a_S **APPLY F**)
Functional set equivalence	 S T $a_S = [v_1..v_N]$ $a_T = [u_1..u_P] = F([v_1..v_N])$ *F* D a_S D a_T a) **N = P** b) **N ≠ P**	a) when **N = P**: (**MAKE** a_T **FROM** (**MAPCAR F ARGS** a_S)) or, as a defered suffix operation: (**MAKE** a_T **FROM** a_S **MAPCAR F**) b) when **N ≠ P**: (**MAKE** a_T **FROM** a_S **MAP** ***type***(a_S) **F**) where the structure of the result set is modified by ***F*** w.r.t the original set (see e.g. the Common LISP function `reduce` -- Steele 1990).

Table 6.4: Representation of the complex attribute level mapping patterns in CSML

Attribute mapping pattern	Schematic presentation of the mapping transformations	CSML specification

Grouping	 S_1 . . . S_n
	T
	$a_{S,1} = v_1$
	$a_{S,N} = v_N$
	a_T = [v_1..v_N]
	D a_S
	D a_T
		a) unconditional, for all **S** , **T**:
		`GROUPS (S FOR ?S)`
		`...`
		`(MAKE aT FROM ?S -> aS)`
		b) conditional, only for some **S** w.r.t. **a**$_S$:
		`(MAKE aT FROM`
		`  (DESCENDANTS S FOR aS`
		`    WHEN cond(S) ) )`

Ungrouping (iteration)	 S
	T_1 . . . T_n
	$a_{T,1} = v_1$
	a_S = [v_1..v_N]
	$a_{T,N} = v_N$
	D a_S
	D a_T
		a) unconditional:
		`(MAKE aT FROM`
		`  (ITERATE-ON aS) )`
		b) conditional:
		`(MAKE aT FROM`
		`  (ITERATE-ON aS`
		`    WHEN cond(S) ) )`

Homo-morphic (1:1 assoc.)	 S
	T
	$a_S = \uparrow R_i$
	$a_T = \uparrow U_i$
	D a_S
	D a_T
	R
	U
		`(MAKE aT FROM`
		`  (ASSOC R REF aS) )`
		where **U** is obtained from the mapping **R** → **U**.
		Because of the one to one correspondence of **R** and **U**, here it is also valid to write:
		`(SAME aT AS aS)`

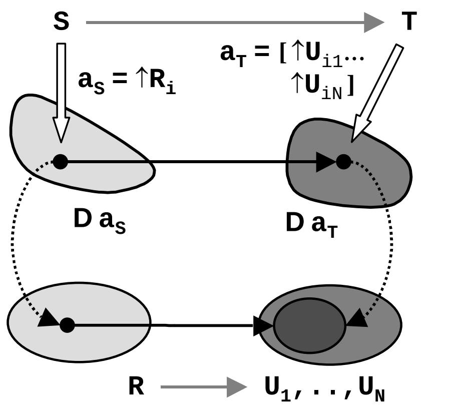
Homo-morphic (1:N assoc.)	 S
	T
	$a_T = [\uparrow U_{i1} \ldots \uparrow U_{iN}]$
	$a_S = \uparrow R_i$
	D a_S
	D a_T
	R
	$U_1, \ldots, U_N$
		`(MAKE aT FROM`
		`  (ASSOC R REF aS) )`
		where **U** is obtained from the mapping **R** → **U**$_1$**...U**$_N$.
		There is no need to express homomorphic 1:N associations in a different way than the 1:1 homomorphic association from above.

Table 6.4 (cont.): Representation of the complex attribute level mapping patterns in CSML

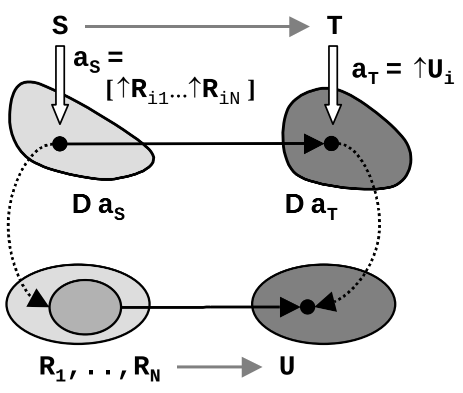
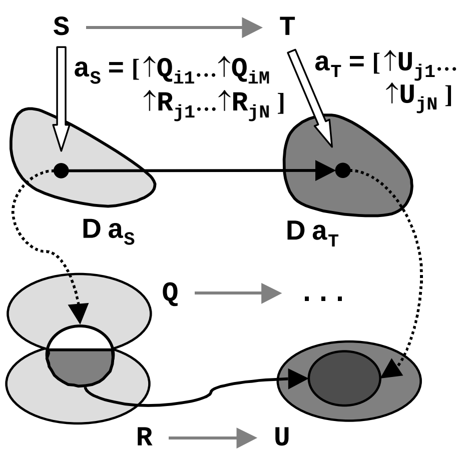
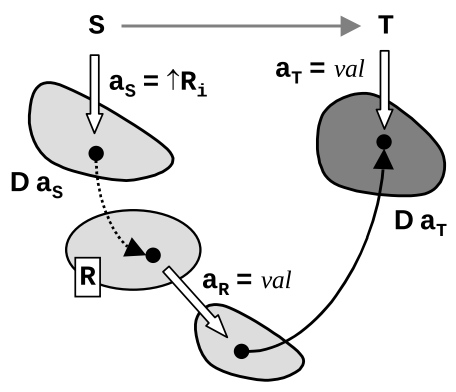
Attribute mapping pattern	Schematic presentation of the mapping transformations	CSML specification
Homo- morphic (N:1 assoc.)	 S T a_S = [↑R_{i1}...↑R_{iN}] a_T = ↑U_i D a_S D a_T R_1, . . , R_N U	a) unconditional: **(MAKE** a_T **FROM** **(ASSOC R REF** a_S**))** b) conditional, only for some **R**: **(MAKE** a_T **FROM** **(ASSOC R REF** a_S **WHEN** ***cond(R)*** **))** where **U** is obtained from the mapping $R_1...R_N \rightarrow U$.
Homo- morphic selective	 S T a_S = [↑Q_{i1}...↑Q_{iM} ↑R_{j1}...↑R_{jN}] a_T = [↑U_{j1}... ↑U_{jN}] D a_S D a_T Q . . . R U	**(MAKE** a_T **FROM** **(ASSOC R REF** a_S**))** Note: In order to accomplish such selective mappings, it is sufficient that *no* reference to the source class **Q** is given. *As it can be seen, all homomorphic associations are achieved by using the same construct (ASSOC). The particular effect depends on the type of the involved data.*
Transitive (telescope)	 S T a_S = ↑R_i a_T = *val* D a_S D a_T R a_R = *val*	**(MAKE** a_T **FROM** **(REF** a_S **FOR** a_R **))** Such references may also be chained: **(MAKE** a_T **FROM** **(REF** a_S **FOR** **(REF . . .** **(REF . . .) . . .** **)** Note: The number of nested REF constructs is theoretically not restricted. However, it is not possible to specify recursively chained pointers with CSML. In such cases, an appropriate functional transformation has to be used.

Table 6.4 (cont.): Representation of the complex attribute level mapping patterns in CSML

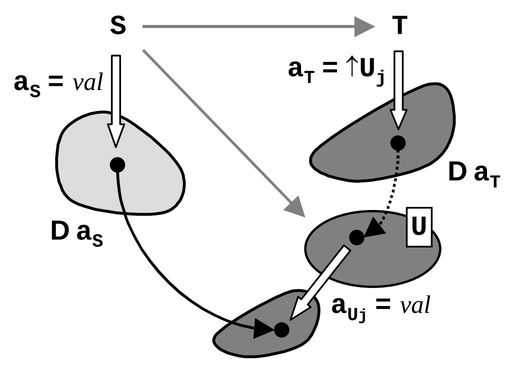
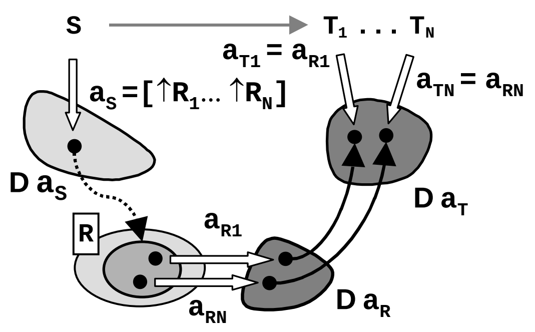
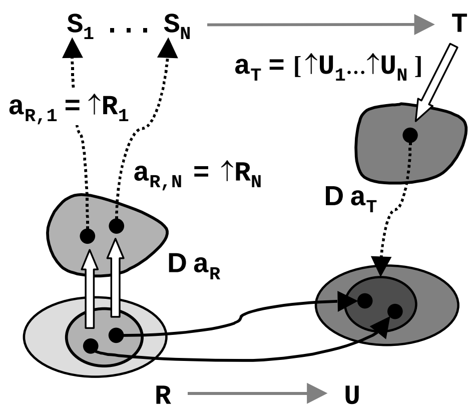
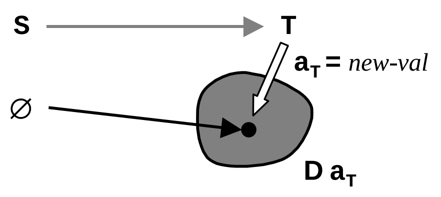
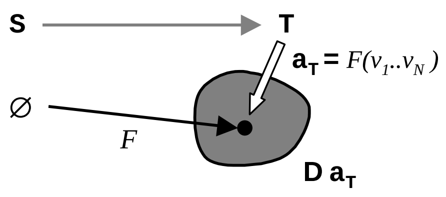
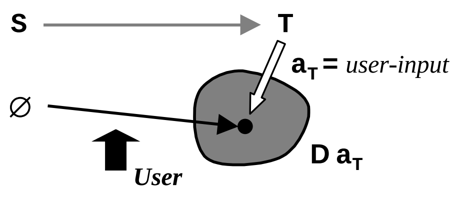
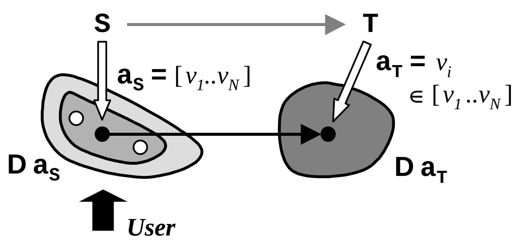
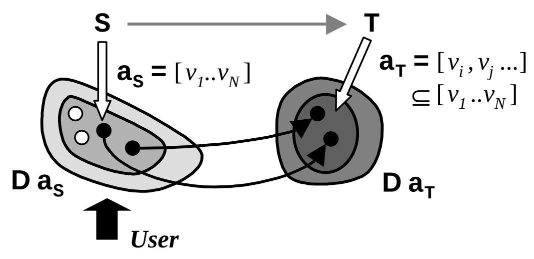
Attribute Mapping Pattern	Schematic presentation of the mapping transformations	CSML specification
Inverse transitive	 S T a_S = *val* $a_T = \uparrow U_j$ D a_T D a_S U a_{Uj} = *val*	**(MAKE a_T FROM (NEW U))** where a_U is obtained from the additional mapping **S → U**, e.g. by: (SAME a_U AS a_S)
Transitive and associative	 S T_1 . . . T_N $a_{T1} = a_{R1}$ $a_S = [\uparrow R_1 \ldots \uparrow R_N]$ $a_{TN} = a_{RN}$ D a_S D a_T R a_{R1} a_{RN} D a_R	**(MAKE a_T FROM** **(ASSOC R REF a_S** **FOR a_R))** Note: In this case the ASSOC template is used in a very similar way as in the set of homomorphic mapping patterns shown above. The only difference is that here it is necessary to use also a FOR clause to point to the required attributes (a_{R1} … a_{RN}).
Inverse associative	 S_1 . . . S_N T $a_T = [\uparrow U_1 \ldots \uparrow U_N]$ $a_{R,1} = \uparrow R_1$ $a_{R,N} = \uparrow R_N$ D a_T D a_R R U	**(MAKE a_T FROM** **(ASSOC U** **WHEN** **(U :=: R)** **(R -> a_R = S -> THIS))** **)** Note: The specification for this mapping pattern is more intricate, partially because it was recognised relatively late, when the syntax of CSML had already been fixed. However, because the target structure is much more comlex than the source, it is by itself a complicated equivalence, requiring to specify explicitly, by means of the idiom (U :=: R), the correspondence of the instances of U and R given in another class mapping declaration.

Table 6.5: Representation of the generative attribute level mapping patterns in CSML

Attribute mapping pattern	Schematic presentation of the mapping transformations	CSML specification
Simple generative		**(MAKE a_T FROM *new-val*)** for example: `(MAKE aT FROM 3.60)`
Functional generative		**(MAKE a_T CONSTRUCTOR F ARGS *v_1 ... v_N*)** for example: `(MAKE absolute_height CONSTRUCTOR z_max_fn ARGS (DESCENDANTS point) ?offset)`
User-dependant generative		**(MAKE a_T FROM (USER-INPUT *type(a_T)* *prompt*))** for example: `(MAKE t_slab FROM (USER-INPUT real "Slab thickness: " ))`
User-dependent selective		**(MAKE a_T FROM (USER-CHOICE ONEOF a_S *prompt*))** for example: `(MAKE ft FROM (USER-CHOICE ONEOF ft_list "Choose found.type" ))`
User-dependent multiple selective		**(MAKE a_T FROM (USER-CHOICE LISTOF a_S *prompt*))** for example: `(MAKE role FROM (USER-CHOICE LISTOF actor_roles "Choose role(s)" ))`

6.5 Representation of Set Relational Operations

Relational database systems are widely used in today's practice and have a solid logical foundation. Since the main goal of a mapping specification is to represent the relationships between a source and a target model which are in a broad sense a superset of the relationships that can exist between relational database tables, it is important to show that all relevant operations defined in relational algebra can be adequately supported.

6.5.1 Basic relational algebra operations

Projection

Let S be a source entity class with attributes $\{ s_i \}_{i=1..n}$, and let T be a target entity class with attributes $\{ t_j \}_{j=1..m} \subset \{ s_i \}$, $m < n$, so that $\forall j \, \exists \, s_{tj} \in \{ s_i \}: s_{tj} \rightarrow t_j$, i.e. $T = \pi_{1..m}(S)$, where $\pi_{1..m}$ is the projection operator.

Then T = **Projection**(S,π) can be achieved in CSML simply by:

```
(MAP CLASS T FROM S
   ATTRIBUTES (SAME t₁ ... tₘ AS sₜ₁ ... sₜₘ)
 )
```

which reduces $\{ s_i \}$ to $\{ t_j \}$ in the target model.

Selection

Let S and T be a source entity class with attributes $\{ s_i \}_{i=1..n}$, and a target entity class with attributes $\{ t_i \}_{i=1..n}$ respectively, so that $\forall \, i \, \exists \, s_i, t_i : s_i \rightarrow t_i$; let also $\{ P_1(s_1 \ ... \ s_m) \ ... \ P_q(s_1 \ ... \ s_m) \}$, $m < n$ be a set of boolean predicates, such that $S_i \rightarrow T_i \Leftrightarrow P_1 \wedge \ ... \ \wedge P_q \, (S_i)$, which is often written as $T = \sigma_{1..q}(S)$, where $\sigma_{1..q}$ is the selection operator.

Then T = **Selection**(S,σ) can be accomplished by:

```
(MAP CLASS T FROM S
   CONDITIONS (PRED P₁ ARGS s₁ ... sₘ) ...
              (PRED P_q ARGS s₁ ... sₘ)
   ATTRIBUTES (SAME ALL)
 )
```

Cartesian product

Let R and S be two source entity classes with attributes $\{ r_i \}_{i=1..n}$ and $\{ s_j \}_{j=1..p}$ respectively, and let T be a target entity class with attributes $\{ t_k \}_{k=1..n+p}$, so that $T = R \times S$.

Then T = **CartesianProduct**(R,S) can be accomplished by:

```
(MAP CLASS T FROM R S
   ATTRIBUTES (SAME t₁ ... tₙ AS r₁ ... rₙ)
              (SAME tₙ₊₁ ... tₙ₊ₚ AS s₁ ... sₚ)
 )
```

Union

Let R and S be two source entity classes with the same arity and attributes { r_i }$_{i=1..n}$ and { s_i }$_{i=1..n}$ respectively, and let T be a target entity class with attributes { t_i }$_{i=1..n}$ so that T = R ∪ S.

Then T = **Union**(R,S) can be accomplished by the two mapping declarations:

```
(MAP CLASS T FROM R
   ATTRIBUTES (SAME t1 ... tn AS r1 ... rn)
 )
```

and

```
(MAP CLASS T FROM S
   ATTRIBUTES (SAME t1 ... tn AS s1 ... sn)
 )
```

Set difference

Let R and S be two source entity classes with key attributes [$r_1 \dots r_k$] and [$s_1 \dots s_k$] respectively, and attributes { r_i }$_{i=1..n}$ of R, $n > k$, and let T be a target entity class so that T = R – S.

Then T = **SetDifference**(R,S,[$r_1 \dots r_k$],[$s_1 \dots s_k$]) can be accomplished by a mapping declaration with a precondition as follows:

```
(MAP CLASS T FROM R S
   GROUPS (S FOR ?S)
   CONDITIONS
     (PRED (LAMBDA (RX SY)
             (NOT (LOOP FOR X IN RX
                        FOR Y IN SY
                        ALWAYS (MEMBER X Y))))
           ARGS (LISTOF r1 ... rk)
                (LISTOF ?S -> s1 ... ?S -> sk))
   ATTRIBUTES
     (SAME t1 ... tn AS r1 ... rn)
 )
```

Obviously, this operation is specified (and performed) by far more difficult than any of the other basic relational operations given above. This is so because, in order to determine the instances of R which do not belong to the set of instances of S, a condition must be provided to check for each key r_k of each instance $R_i \in R$ whether it is not a key of an instance $S_j \in$ { S } $_{j=1..s}$. To achieve that, an in-line Common LISP function is defined, i.e. (LAMBDA (RX SY) ...). This function works on two arguments: the list of keys [$r_1 \dots r_k$] of the instance of R to be checked, and a list of lists of the keys [$s_1 \dots s_k$] of all instances of S. The function returns *true* if r_i is not found in S, and *false* otherwise. The second argument to the function is provided by grouping all instances of S in a variable ?S, and then using this variable together with the term operator LISTOF, to produce the required list [[$s_{11} \dots s_{k1}$] ... [$s_{1s} \dots s_{ks}$]]. By a non LISP implementation of CSML this function would need to be replaced by an external function.

6.5.2 Higher order relational algebra operations

All higher order relational algebra operations can be derived easily from the five basic relational operations given in the previous section (cf. Ullman, 1988). This is valid also for the respective mapping specifications. However, as in RDBMS, a “direct” specification taking into account the specific character of the respective higher order relational operation is more efficient in each particular case.

Intersection

An intersection is equivalent to two consecutive *SetDifference* operations.

Let R and S be two source entity classes with key attributes $[\ r_1 \dots r_k\ ]$ and $[\ s_1 \dots s_k\ ]$ respectively, and attributes $\{\ r_i\ \}_{i=1..n}$ of R, $n > k$, and let T be a target entity class so that $T = R \cap S$.

Then T = **Intersection**$(R,S,[\ r_1 \dots r_k\ ],[\ s_1 \dots s_k\ ])$ can be accomplished by:

```
(MAP CLASS T FROM R S
   CONDITIONS
     (r₁ = s₁) ... (rₖ = sₖ)
   ATTRIBUTES
     (SAME t₁ ... tₙ AS r₁ ... rₙ)
)
```

Having in mind that $T = R \cap S = R - (R - S)$, **Intersection**(...) can also be achieved by declaring an additional variable ?S, so that ?S = **SetDifference**(...), and then applying this variable in the mapping $R \rightarrow T$, where ?S plays exactly the role of the grouped instances given by GROUPS (S FOR ?S) in the definition of the **SetDifference** operation. However, it is obvious that this would be by far less efficient than the direct transformation $R,S \rightarrow T$ shown above.

Join

A join is equivalent to a cartesian product combined with a condition.

Thus, given that R and S are two source entity classes with attributes $\{\ r_i\ \}_{i=1..n}$ and $\{\ s_j\ \}_{j=1..p}$ respectively, and $P_\theta\ (\rho,\sigma)$ is a predicate on $\rho \in \{\ r_i\ \}$, $\sigma \in \{\ s_j\ \}$ so that $R \times S \xrightarrow[P_\theta]{} T$, which is usually written as $T = R \underset{\rho P_\sigma}{\bowtie} S$,

T = **Join**$(R,S,P_\theta(\rho,\sigma))$ can be accomplished by:

```
(MAP CLASS T FROM R S
   CONDITIONS
     (PRED Pθ ARGS R -> ρ S -> σ)
   ATTRIBUTES
     (SAME t₁ ... tₙ AS r₁ ... rₙ)
     (SAME tₙ₊₁ ... tₙ₊ₚ AS s₁ ... sₚ)
)
```

Equijoin

An equijoin is a join with a condition $P_e : (\rho = \sigma)$.

$T =$ **EquiJoin**(R, S, P_e) can be specified simpler than a join as follows:

```
(MAP CLASS T FROM R S
   CONDITIONS (ρ = σ)
   ATTRIBUTES
     (SAME t1 ... tn AS r1 ... rn)
     (SAME tn+1 ... tn+p AS s1 ... sp)
 )
```

Natural join

A natural join is effectively a join with a condition specified as equality over a set of key attributes in two source entities.

Thus, given the source entity classes R and S with key attributes $[\ r_1 \dots r_k\ ]$ and $[\ s_1 \dots s_k\ ]$ respectively, and the condition $P_k : r_1 = s_1 \wedge \dots \wedge r_k = s_k\ (R \times S)$, and given a target entity class T, so that $T = R \bowtie S$, $T =$ **NaturalJoin**(R, S, P_k) can be accomplished by:

```
(MAP CLASS T FROM R S
   CONDITIONS (r1 = s1) ... (rk = sk)
   ATTRIBUTES
     (SAME t1 ... tn AS r1 ... rn)
     (SAME tn+1 ... tn+p AS s1 ... sp)
 )
```

Semijoin

A semijoin is a projection onto the attributes of R of the natural join of two source entities R and S, i.e. $T = R \ltimes S = \pi_R(R \bowtie S)$.

$T =$ **SemiJoin**(R, S, P_k, π_R) can be achieved with the same specification as for natural join, leaving out the last attribute mapping, i.e. (SAME $t_{n+1} \dots t_{n+p}$ AS $s_1 \dots s_p$).

6.5.3 Nested operations

Unlike SQL, CSML does not support nested class mapping declarations and consequently cannot combine relational operations explicitly. However, noticing that free variables may be used not only as a source, but also as the target of a mapping, it is possible to link relational operations implicitly. To achieve this, temporary variables should be used for any nested operations; it is only necessary that all intermediate structures are defined in the target model, but this is not a serious restriction for the envisaged use of CSML in an IFC-based CEE system.

For example, assume that $T =$ **Projection**(S, π). Then $U =$ **Selection** $($**Projection**$(S, \pi), \sigma)$ can be accomplished by:

```
(MAKE VAR ?X FROM T -> THIS)      -- initialisation of an empty global variable
...                                  of type T
(MAP CLASS ?X FROM S ... )        -- projection (as on page 183)
...
(MAP CLASS U FROM ?X ... )        -- selection (as on page 183).
```

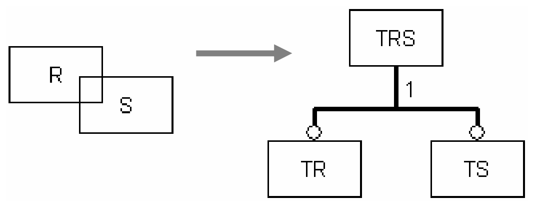
6.5.4 View Integration Operations

As already mentioned in chapter 5, an important issue in the conceptual design of federated databases is the derivation of an integrating database schema for a set of partially harmonised, overlapping databases. The efforts in this area have helped to identify a number of high-level operations that are interesting also for the model transformation problems in CEE. Therefore, to complete the proof for sufficiency of CSML, a subset of the view integration operations proposed initially in (Motro 1987) and anticipated as relevant to different mapping tasks is analysed below.

Meet

A *meet* expresses the mapping of two overlapping source model classes with a common key but no common superclass to three target model classes, a superclass and two non overlapping subclasses, as illustrated on fig. 6.5.

```
ENTITY R;
  A : NUMBER;
  B : STRING;
END_ENTITY;
ENTITY S;
  A : NUMBER;
  C : REAL;
END_ENTITY;
```



```
ENTITY TRS;
  A : NUMBER;
END_ENTITY;
ENTITY TR
  SUBTYPE OF (TRS);
  B : STRING;
END_ENTITY;
ENTITY TS
  SUBTYPE OF (TRS);
  C : REAL;
END_ENTITY;
```

Fig. 6.5: Schematic representation of a “Meet”

Thus, if R and S are two source entity classes with respective attributes $\{ r_i \}_{i=1..n}$ and $\{ s_j \}_{j=1..p}$, a common key $[r_1 \ldots r_k] = [s_1 \ldots s_k]$, and shared attributes $[r_1 \ldots r_m]$ and $[s_1 \ldots s_m]$, where $p > n > m > k$; and T_R, T_S and T_{RS} are three target entities, so that $R \rightarrow T_R$, $S \rightarrow T_S$ and T_{RS} is a generalisation of T_R and T_S, then $\{ T_R, T_S, T_{RS} \}$ = **Meet**(R,S) can be accomplished by:

(MAP CLASS T_R FROM R
ATTRIBUTES (SAME $t_{R1} \ldots t_{Rn}$ AS $r_1 \ldots r_n$))
(MAP CLASS T_S FROM S
ATTRIBUTES (SAME $t_{S1} \ldots t_{Sp}$ AS $s_1 \ldots s_p$))

T_{RS} must not be mapped because both T_R and T_S inherit from it, but there is no equivalent superclass in the source model.

Fold

A *fold* combines two source classes standing in sub/super class relationship to each other to a single target class encompassing both.

Let R and S be two source entity classes with attributes $\{ r_i \}_{i=1..n}$ and $\{ s_j \}_{j=1..p}$ respectively, where R is a generalisation of S, and let T be a target entity class, such that $\{ R \} \subset \{ T \} \wedge \{ S \} \subseteq \{ T \}$.

T = **Fold**(R,S) can be accomplished as follows:

```
(MAP CLASS T FROM R
   ATTRIBUTES (SAME t1 ... tn AS r1 ... rn))
(MAP CLASS T FROM S
   ATTRIBUTES (SAME t1 ... tp AS s1 ... sp))
```

The inverse case T → R,S is more difficult. It needs the introduction of 2 additional temporary variables (?T), along with the use of knowledge-based templates:

```
(MAP CLASS S FROM T
   VAR (MAKE ?T FROM T -> THIS)
   CONDITIONS (PRED (AND (?T HAS tn+1) ... (?T HAS tp)))
   ATTRIBUTES (SAME s1 ... sp AS t1 ... tp))
(MAP CLASS R FROM T
   VAR (MAKE ?T FROM T -> THIS)
   CONDITIONS (PRED (NONE (?T HAS tn+1) ... (?T HAS tp)))
   ATTRIBUTES (SAME r1 ... rn AS t1 ... tn))
```

Aggregation

An *aggregation* transforms a source entity having an attribute with a complex data type (set, list) into a target entity linked to a set of entities each having an attribute equivalent to the respective element in the set or list of values of the complex attribute of the source entity. Thus, an aggregation is in effect the "entity level" version of the iterative (ungrouping) mapping pattern given in table 6.4.

Let S be a source class with attributes { s_1 ... s_t ... s_n } where s_t = [t_1 ... t_p], and t_1 ... t_p are all of type θ. Further, let T be a target class with one attribute t of type θ, and Ta be a target class with attributes { t_{a1} ... t_{at} ... t_{an} } such that $\forall$ i $\exists$ S_i,Ta_i : S_i → Ta_i $\wedge$ [t_1 ... t_p] → [T_1 ... T_p].

Then { T, Ta } = **Aggregation**(S,s_t) can be achieved as follows:

```
(MAP CLASS T FROM S
   ATTRIBUTES
     (MAKE t FROM (ITERATE-ON st)))
(MAP CLASS Ta FROM S
   ATTRIBUTES
     (SAME ta1 ... tat-1 tat+1 ... tan AS s1 ... st-1 st+1 ... sn)
     (MAKE tat FROM (NEW T)))
```

Here, the first class mapping applies an iterator over the list s_t creating a separate entity for each t_i in s_t, and the second class mapping uses the new entity instances of T, associating them with Ta.

Inversely, for Ta,T → S, the attribute s_t is created by constructing the list of all attributes t of the instances of T referenced by Ta. This is much simpler (and faster):

```
(MAP CLASS S FROM Ta T
   ATTRIBUTES
     (SAME s1 ... st-1 st+1 ... sn AS ta1 ... tat-1 tat+1 ... tan)
     (MAKE st FROM (REF tat FOR t)))
```

Telescoping

Telescoping involves a set of *transitive* mappings (see table 6.4).

Let R and S be two source entity classes with attributes { r_i ... r_n r_S } and { s_i }$_{i=1..p}$ respectively, where r_S = ↑s_i | s_i ∈ { S }.

Then T = **Telescope**(R,S,r_S) is achieved simply by:

```
(MAP CLASS T FROM R
   ATTRIBUTES
     (SAME t1 ... tn AS r1 ... rn)
     (MAKE tn+1 FROM (REF rS FOR s1))
     ...
     (MAKE tn+p FROM (REF rS FOR sp)))
```

The inverse case, T → R,S,r_s would involve several NEW operators and may not always be possible to achieve.

Is-a Join

An Is-a Join transforms two overlapping source classes into a single class in the target model representing their joined properties, i.e. their *generalisation*.

Let R and S be two source entity classes with attributes { r_i }$_{i=1..n}$ and { s_j }$_{j=1..p}$ respectively, which share a common key [r_1 ... r_k] = [s_1 ... s_k], $n > k$, $p > k$; and let T be a target entity class which is defined as a generalisation of R and S.

Then T = **Is_a_Join**(R,S,[r_1 ... r_k],[s_1 ... s_k]) can be achieved by:

```
(MAP CLASS T FROM R S
   CONDITIONS (r1 = s1) ... (rk = sk)
   ATTRIBUTES
     (SAME t1 ... tn AS r1 ... rn)
     (SAME tn+1 ... tn+p-k AS sk+1 ... sp))
```

In (Motro 1987) this operation is called simply a “*join*”. I prefer the name “*Is-a Join*” to distinguish it from the standard relational operation with the same name, and also because I find “*Is-a Join”* a more suitable name for the addressed issue.

Addition, Deletion and Renaming

These last three trivial operations complete the set of integration operations relevant to mapping problems. They can be realised as follows.

The **addition** of one attribute t_a to the target entity class T which is otherwise identical to the source class S can be accomplished e.g. by:

```
(MAP CLASS T FROM S
   VAR (MAKE ?A CONSTRUCTOR F ARGS X Y ... )
   ATTRIBUTES
     (SAME ALL)
     (MAKE ta FROM ?A))
```

where the constructor function F performs the necessary initialisation for t_a.

The **deletion** of one attribute s_a of S in T is done by copying all attributes of S except for s_a:

```
(MAP CLASS T FROM S
   ATTRIBUTES (SAME ALLBUT sa ))
```

At last, **renaming** of S to T in the target model without changing any attribute values can be done simply by:

```
(COPY CLASS T FROM S)
```

Table 6.6 below summarises the anticipated use, the specification efforts and the run-time performance of the discussed relational operations based on the modest experience and judgement of the author gathered from performed test studies.

Table 6.6: Summary of relational operations usage in mapping problems

Operation	Anticipated use in model mapping	Representation with CSML	Run-time performance in prototype environment using CSML
Projection	common	easy	fast
Selection	common	easy	depends on the conditions, mostly moderate performance
Cartesian product	moderate	easy	slow (depends heavily on the number of instances)
Union	rare	easy	fast
Set difference	rare	complicated	moderate / slow
Intersection	rare	easy	moderate / slow
Join	common	easy	moderate to slow, depending on the conditions
Equijoin	common	easy	moderate
Natural join	common	easy	moderate / slow
Semijoin	moderate	easy	moderate / slow
Meet	common	easy	moderate
Fold	common	easy, but complicated inverse case	moderate
Aggregation	common	medium difficulty	moderate / slow
Telescoping	common	easy; inverse case not always possible	fast
Is-a join	moderate	easy	moderate / slow
Addition	common	medium difficulty	depends on the constructor function
Deletion	common	easy	fast
Renaming	common	easy	fast

Knowing these qualitative measures and the respective mapping patterns, it should not be difficult to estimate the development efforts and the expected performance of a specific mapping task before undertaking the effort of its detailed realisation.

6.6 Typical Mapping Examples

The following illustrative examples are adapted from publications describing other known mapping approaches (Clark 1992; Hardwick 1994; Bailey 1995; Amor 1997). As the main goal of CSML is to facilitate run-time mappings between shared model repositories and local domain or application-specific models, the examples are selected so that the representational power of CSML in such kinds of problems is demonstrated. For conciseness, basically only the mapping from an assumed shared source model to a local target model is given, although in some selected cases the inverse mapping is shown as well. The keyword `SHARED` designating integrated, shared models in the system is omitted in all examples except for example 1, as it does not contribute much to the illustration of the separate mapping concepts. The existence of a shared model is not mandatory for CSML anyway.

Example 1: *adapted from (Clark 1992)*

This example shows the *mapping between two conceptually different representations* of a 2-dimensional point. The same procedure can be readily extended for other typical resource entities used in building product data models.

Source schema →	***Target schema***

`SCHEMA core_model;`
`ENTITY point;`
`  r, theta : REAL;`
`END_ENTITY;`
`END_SCHEMA;`
	`SCHEMA local_model;`
	`ENTITY point;`
	`  x, y : REAL;`
	`END_ENTITY;`
	`END_SCHEMA;`

Mapping specification:

```
(MAP local_model FROM SHARED core_model
 CLASSES
  (MAP CLASS local_model:point FROM core_model:point
    ATTRIBUTES
      (MAKE x FROM (APPLY (LAMBDA (RS TS) (* RS (COS TS)))
                          ARGS r theta))
      (MAKE y FROM (APPLY (LAMBDA (RS TS) (* RS (SIN TS)))
                          ARGS r theta))
    )
 )
```

Inverse mapping:

```
(MAP SHARED core_model FROM local_model
 CLASSES
  (MAP CLASS core_model:point FROM local_model:point
    ATTRIBUTES
      (MAKE r FROM (APPLY (LAMBDA (X Y) (SQRT (+ (* X X) (* Y Y))))
                          ARGS x y))
      (MAKE theta FROM (APPLY (LAMBDA (X Y) (TAN (/ Y X)))
                              ARGS x y))
    )
 )
```

Comment:

In this simple example both forward and inverse mapping from/to a shared model to/from a view (or application-specific) model are demonstrated. Although the mapping contains only trivial transformation formulae, I nevertheless prefer the given separate mapping definitions for the two mapping directions compared to the bi-directional mapping, advocated e.g. in (Amor 1997). Separate mapping definitions provide better conceptual clarity and, in general, can be easier implemented. Besides this, I doubt that in larger models it would always be possible to provide symmetric equivalence specifications.
Most interesting in the given example is the use of `LISP` forms for the geometric transformations:

```
   x = f(r,θ) = r * cosθ;
   y = f(r,θ) = r * sinθ.
```

The local scope of the variables in each `LISP` form can be clearly recognised in the last two attribute mappings (the parameters `X` and `Y` in the given `LAMBDA` functions are not the same as the attributes `x` and `y` in the attribute mapping specification, but are substituted for the actual values of these attributes upon execution).
The schema names in the class mapping declarations are used purely to emphasise the direction of the mapping. They are not needed to distinguish the source and target entities here (both with name `point`), as they are identifiable by position.

Example 2: *adapted from (Bailey 1995)*

This example shows a simple mapping between two *structurally similar* representations of a circle. It is included mainly to emphasise the difference to example 3.

Source schema →	***Target schema***

`SCHEMA shared_model;`
`ENTITY circle;`
`  centre : point;`
`  radius : REAL;`
`END_ENTITY;`
`ENTITY point;`
`  x, y, z : REAL;`
`END_ENTITY;`
`END_SCHEMA;`
	`SCHEMA local_model;`
	`ENTITY circ;`
	`  centre   : point;`
	`  diameter : REAL;`
	`END_ENTITY;`
	`ENTITY point;`
	`  x_coord  : REAL;`
	`  y_coord  : REAL;`
	`  z_coord  : REAL;`
	`END_ENTITY;`
	`END_SCHEMA;`

Mapping specification:

```
(MAP local_model FROM shared_model
 CLASSES
  (MAP CLASS circ FROM circle
    ATTRIBUTES
      (SAME centre)
      (MAKE diameter FROM radius APPLY (LAMBDA (X) (* X 2.0))) )
 DEPENDENT-CLASSES
  (MAP CLASS point FROM point
    ATTRIBUTES (SAME x_coord y_coord z_coord AS x y z) )
 )
```

Comment:

Most interesting here is the use of the `DEPENDENT-CLASSES` specification section. According to the given definitions, instances of `point` will be included in the target model if and only if they are referenced by at least one other entity instance (in this case `circle`). In contrast, if the mapping specification for `point` had been included in the `CLASSES` section, all points contained in the source model would have been mapped to the target, regardless of whether they are associated to circle entities or not.

The example shows also the easy treatment of synonyms with the help of the `SAME` construct.

Example 3: *adapted from (Bailey 1995)*

This example shows the mapping for *structurally dissimilar* representations of a circle.

Source schema →	***Target schema***
`SCHEMA shared_model;`	`SCHEMA local_model;`
`ENTITY circle;`	`ENTITY circ;`
`  radius     : REAL;`	`  centre   : point;`
`  cx, cy, cz : REAL;`	`  diameter : REAL;`
`END_ENTITY;`	`END_ENTITY;`
`END_SCHEMA;`	`ENTITY point;`
	`  x, y, z  : REAL;`
	`END_ENTITY;`
	`END_SCHEMA;`

Mapping specification:

```
(MAP local_model FROM shared_model
 CLASSES
  (MAP CLASS point FROM circle
    ATTRIBUTES (SAME x y z AS cx cy cz) )
  (MAP CLASS circ FROM circle
    ATTRIBUTES
      (MAKE diameter FROM radius
                     APPLY (LAMBDA (X) (* X 2.0)))
      (MAKE centre FROM (NEW point))
    )
 )
```

Inverse mapping:

```
(MAP shared_model FROM local_model
 CLASSES
  (MAP CLASS circle FROM point circ
    ATTRIBUTES
      (MAKE radius FROM diameter APPLY (LAMBDA (X) (/ X 2.0)))
      (MAKE cx cy cz FROM (REF centre FOR x y z))
    )
 )
```

Comment:

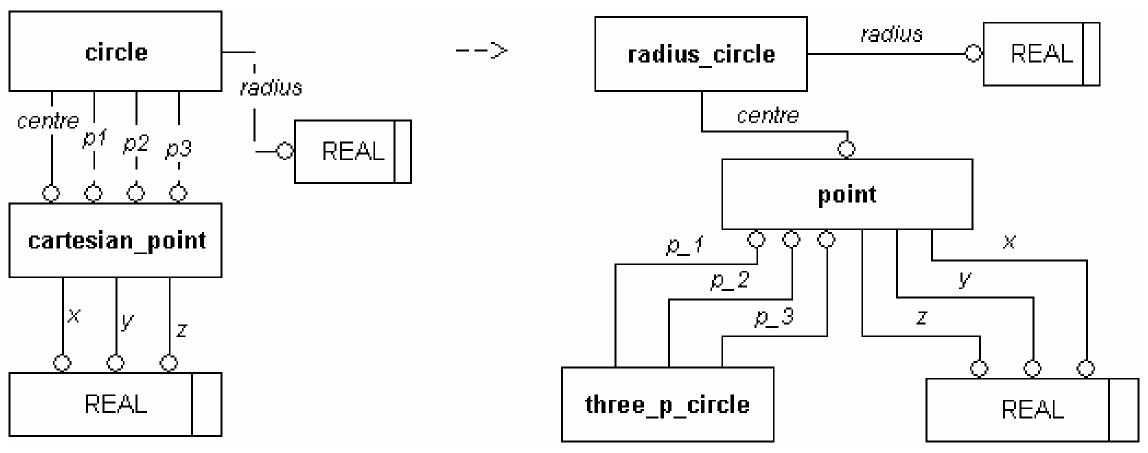
Here a typical case of 1:C mapping is shown. In the forward mapping, which presents the more difficult case, the attribute `centre` of `circ` is assigned a pointer to the newly created `point` instances according to the first `MAP CLASS` declaration. Bailey and Amor, who uses the same example for VML (Amor 1997), claim that *invariants* have to be specified here to provide the requisite information needed to create a `point` for each `circle` when mapping from `shared_model` to `local_model`. In CSML this is not necessary. The `circle` entity is supposed to maintain a link to all entities created by its mapping, which allows to associate `circ` and `point` entities unambiguously during the mapping process. In order to express such links, the `NEW` operator has been introduced.
The inverse mapping demonstrates how references between entities can be traversed to obtain the needed data.

Example 4: *adapted from (Bailey 1995)*

This example shows the tackling of a typical *classification problem.*

Source schema →	***Target schema***
`SCHEMA shared_model;`	`SCHEMA local_model;`
`ENTITY circle;`	`ENTITY point;`
`  centre   : cartesian_point;`	`  x, y, z : REAL;`
`  radius   : OPTIONAL REAL;`	`END_ENTITY;`
`  p1 : OPTIONAL cartesian_point;`	`ENTITY radius_circle;`
`  p2 : OPTIONAL cartesian_point;`	`  centre  : point;`
`  p3 : OPTIONAL cartesian_point;`	`  radius  : REAL;`
`END_ENTITY;`	`END_ENTITY;`
`ENTITY cartesian_point;`	`ENTITY three_p_circle;`
`  x, y, z  : REAL;`	`  p_1, p_2, p_3 : point;`
`END_ENTITY;`	`END_ENTITY;`
`END_SCHEMA;`	`END_SCHEMA;`

EXPRESS-G:



Source schema Target schema

Mapping specification:

```
(MAP local_model FROM shared_model
 CLASSES
  (COPY CLASS point FROM cartesian_point)
  (MAP CLASS radius_circle FROM circle
    CONDITIONS (PRED (radius OF circle))
    ATTRIBUTES (SAME centre radius)
    )
  (MAP CLASS three_p_circle FROM circle
    CONDITIONS (PRED (AND (p1 OF circle) (p2 OF circle) (p3 OF circle)))
    ATTRIBUTES (SAME p_1 p_2 p_3 AS p1 p2 p3)
    )
 )
```

Comment:

The single representation of `circle` in the source schema allows for different representation options that have to be mapped to different entities in the target: `radius_circle` and `three_p_circle`. Both mapping declarations include a condition using as predicate the knowledge-based attribute existence template `(attr OF instance)` to discriminate between circles defined by three points and circles defined by centre point and radius. The attribute mappings themselves are quite trivial, and are easily achieved with the help of the SAME construct. The last line shows also one typical case for the use of parallel lists.

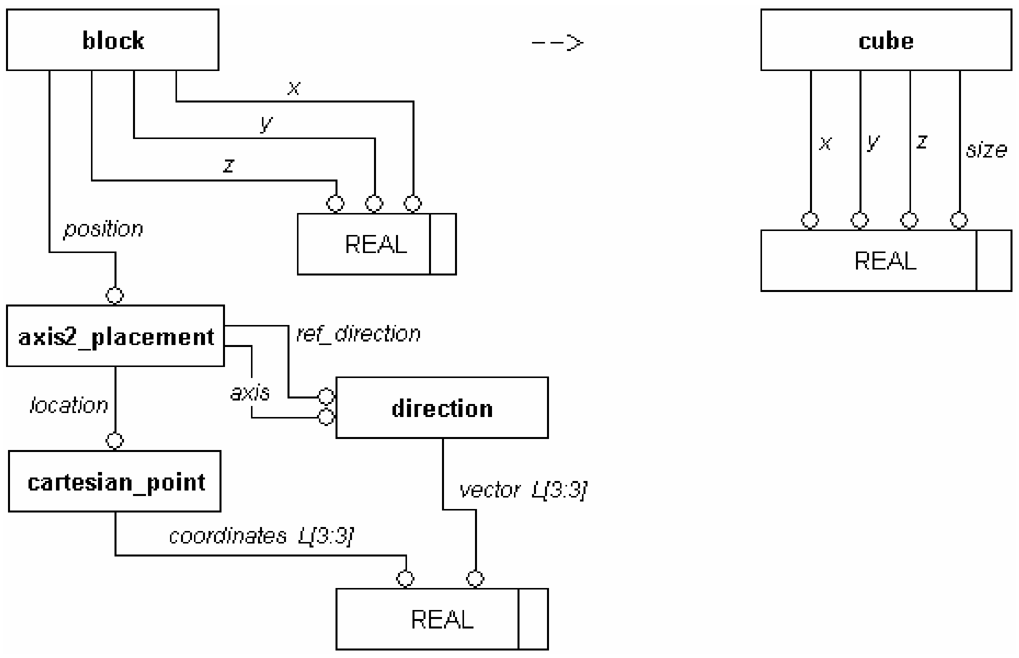
Example 5: *adapted from (Hardwick 1994)*

This example shows a mapping that *reduces the representation* of a 'block' entity as defined in the STEP AP 203 (ISO 10303-203 1994) to a much simpler structure in the target model. Such mappings are typical for the derivation of application-specific views from a common shared model.

Source schema →	***Target schema***

`SCHEMA AP_203;`
`ENTITY block;`
`  position : axis2_placement;`
`  x, y, z  : REAL;`
`END_ENTITY`
`ENTITY axis2_placement;`
`  axis : direction;`
`  ref_direction : direction;`
`  location : cartesian_point;`
`END_ENTITY;`
`ENTITY direction;`
`  vector : LIST [3:3] OF REAL;`
`END_ENTITY;`
`ENTITY cartesian_point;`
`  coordinates : LIST [3:3] OF REAL;`
`END_ENTITY;`
`END_SCHEMA;`
	`SCHEMA view_model;`
	`ENTITY cube;`
	`  x, y, z : REAL;`
	`  size    : REAL;`
	`END_ENTITY;`
	`END_SCHEMA;`

EXPRESS-G:



Source schema Target schema

Mapping specification:

```
(MAP view_model FROM AP_203
 CLASSES
  (MAP cube FROM block
    CONDITIONS
      (block -> x = block -> y)
      (block -> x = block -> z)
    ATTRIBUTES
      (SAME size AS x)
      (MAKE x y z
            FROM (REF position
                      FOR (REF location
                               FOR coordinates)) )
    )
 )
```

Comment:

In the source model `x`, `y` and `z` represent the dimensions of a *block*, whereas in the target model the same attributes stand for the coordinates of a *cube*'s origin w.r.t. the global coordinate system.

The mapping is performed only for *block* instances with equal dimensions. The "filtering" conditions `x = y` and `x = z` require full specification of the attributes `x`, `y`, `z` in the form `class -> attribute` because the same names exist in both schemas and would not otherwise be properly distinguished.

The REF clause in the last attribute mapping statement shows a typical *telescope* case together with *list destructuring*[*)], i.e. `∀ b ∈ { block } ( b.x = b.y = b.z ) :`
`b.position = ↑a ∧ a.location = ↑p ∧ p.coordinates → [ c.x c.y c.z ]`
where: `a ∈ { axis2_placement }, p ∈ { cartesian_point }, c ∈ { cube }.`

Example 6: *adapted from (Amor 1997)*

This last example presents a case of a *model transformation which cannot be tackled automatically in both directions* (in the inverse mapping, height and width cannot be calculated from the existing data in the local model).

Source schema →	***Target schema***
`SCHEMA shared_model;`	`SCHEMA local_model;`

`ENTITY wall;`
`  height, width : REAL;`
`END_ENTITY;`
	`ENTITY wall;`
	`  area : REAL;`
	`END_ENTITY;`

`END_SCHEMA;`	`END_SCHEMA;`

Mapping specification:

```
(MAP local_model FROM shared_model
 CLASSES
  (MAP wall FROM wall
    ATTRIBUTES
      (MAKE area FROM
                 (APPLY (LAMBDA (X Y) (* X Y))
                        ARGS height width))
    )
 )
```

Inverse mapping:

```
(MAP shared_model FROM local_model
 CLASSES
  (MAP wall FROM wall
    ATTRIBUTES
      (MAKE height FROM (USER-INPUT
                           REAL "Height of wall ~: " OBJECTNAME))
      (MAKE width FROM (APPLY (LAMBDA (A H) (/ A H))
                               ARGS area (NEW wall FOR height)))
    )
 )
```

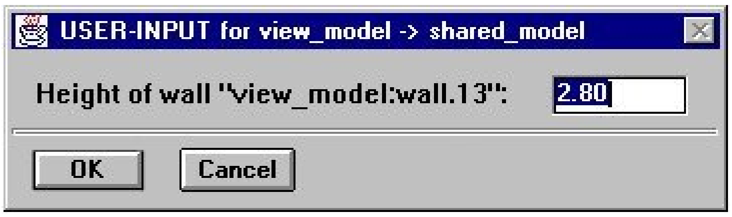
[*)] *List destructuring* is a powerful feature of CSML, but it must be applied with care as it is at the sole responsibility of the developer of the mapping specification to ensure that a list mapped to individual items will contain correct elements w.r.t. their number and data types. It is not required that all elements in such lists are of the same type, important is the *type correspondence*.

Comment:

The forward mapping is fairly simple here; it is in principle very similar to the mapping shown in example 1. The values for the attribute `area` are obtained by using the APPLY operator with a LAMBDA function which defines the required functional relationship, i.e. `area = height * width`.

The inverse mapping is less obvious. As already mentioned, `local_model` does not contain all necessary data for a transformation back to `shared_model`. This means that the mapping from `local_model` to `shared_model` cannot be performed automatically, but it should *not* mean that it cannot be performed at all. In many such cases the complete data set needed for the model transformations can in fact be obtained with only a few additional, ad hoc provided data. Such data can be specified with the help of the constructs USER-INPUT and USER-CHOICE. In the above example, USER-INPUT will give a value to the `height` attribute of the target entity `wall`, after which its `width` can be easily deduced.

Amor does not explain how such situations are treated in VML, and in other known mapping approaches there does not seem to be paid enough attention to this issue either. However, whilst it is certainly correct to conclude that a mapping with insufficient source model data is in general not possible, it is not satisfactory to give up the whole procedure only because of a few missing data, especially when it is not very difficult for the user to fill in the gaps. The pragmatic approach taken in CSML helps solving such problems. Of course, the use of interactive mapping constructs like USER-INPUT and USER-CHOICE would require an appropriate application-side implementation, but this is not very difficult to achieve. For example, a simple dialog box can be implemented for USER-INPUT, as shown on the sample screenshot below, and a list selection can be implemented for USER-CHOICE (in both cases accompanied by an appropriate graphical presentation for user convenience). The expected data should be requested by the mapping engine after parsing the mapping specifications, but before the actual mapping procedure starts. The mapping is given up only if the user does not provide all requested data, there are no default values specified, and the mode of the mapping operation is set to "*complete*".


USER-INPUT for view_model -> shared_model

Height of wall "view_model:wall.13": 2.80

Fig. 6.6: Sample client implementation for the USER-INPUT mapping construct

6.7 Discussion

The formal specification of mappings is an essential step in the design of an interoperable system for concurrent engineering. With CSML the features needed to enable successful application of the model mapping approach are provided in the following manner:

1) *Semantic and descriptive conflicts* are resolved with the help of a rich set of mapping constructs on entity and attribute level, including the treatment of synonyms and homonyms, value type and value range conflicts etc. Of course, as in all other

approaches, these constructs provide only recipes how to resolve such conflicts; their recognition must be done manually by the developer of the mapping specification.
Unit and scalability conflicts are not supported explicitly in CSML. However, scalability should not be a problem if the implemented mapping engine works with sufficient internal precision. Unit handling can be achieved easily with the available language constructs for all models that utilise the integrated resources of STEP providing formal specifications for “measure values” and “units of measure” (see ISO 10303-41 1994). This is granted by default by all STEP-based models as well as by the IFC framework. On the other hand, if a specific application model does not define explicit measures for physical quantities, there will be no way to define unit conversion except by external functions; and if both involved models do not support conceptually unit definitions, there will be no way to do this at all. In such cases, implicit assumptions have to be agreed[*)].

2) *Structural conflicts* are supported for all identified mapping patterns, as well as for all known operations from RDBMS. Table 6.7, re-drawn from the principal presentation of structural mapping types given in table 5.1, provides a summary of the capabilities of CSML in that respect.

Table 6.7: Supported mapping types in CSML

Cardi-nality	**Class → Class**	**Entity → Entity Inst. Inst.**	**Attr.→ Attr.**	**Entity → Attr. Inst.**	**Attr. → Entity Inst.**
`0 : 1`	N/A	▨ **implicit**	▨ **+**		
`1 : 0`	▨ **implicit**	▨ **implicit**	▨ **implicit**		
`1 : 1`	▨ **+**	▨ **+**	▨ **+**	▨ **+**	▨ **+**
`1 : C`	▨ **+**	▨ **+**	▨ **+**	▨ **+**	▨ **+**
`C : 1`	▨ **+**	▨ **+**	▨ **+**	▨ **+**	▨ **+**
`1 : N`		▨ **+**			▨ **+**
`N : 1`		▨ **+**		▨ **+**	
`C : B`	**+**	**+**	▨ **+**	▨ **supported by ext. functions**	▨ **supported by ext. functions**
`C : N`		**+**			▨ **supported by ext. functions**
`N : C`		**+**		▨ **supported by ext. functions**	
`N : M`		**(+)**			

(**Note**: hatched boxes = mapping types relevant to the IFC modelling framework)

[*)] The problem of unit handling for physical quantities in engineering data models has been analysed in detail in (Gruber 1993) who shows an example ontology addressing measure-related issues. The only language that can deal with such ontological specifications provided in the body of a mapping specification is ACL/KIF (Genesereth & Fikes 1992).

In the above table, M:N mappings are given in parenthesis because currently they are still not sufficiently tested. However, for the modelling framework of IAI/IFC, which constrains the EXPRESS paradigm with additional restrictive rules, this mapping type is actually not needed. More difficult to implement are only the multiple cardinality constructs "entity instance to attribute" and vice versa requiring in many of the examined cases the use of external functions.

3) *Heterogeneity conflicts*, i.e. the use of different representation paradigms, have not been explicitly considered in the scope of CSML. However, CSML is not specifically bound to EXPRESS and can be readily adapted for other object-oriented models. As demonstrated in section 6.5, the mapping of relational models is covered as well.

4) *The design of the language* itself follows the requirements identified in section 5.6.5. Great attention has been paid to the requirements for modularity and the use of a declarative style for the mapping specifications so that an adaptation to different implementation environments should not be difficult to achieve.

   The use of templates, which is a novel feature of CSML, gives greater transparency and allows for more accurate estimation of the complexity of each particular mapping task. Besides, as self-contained constructs, templates provide an easy way to extend the scope of CSML in case that further mapping patterns are going to be needed.

   The language syntax is intentionally not aligned with EXPRESS, even though at several places certain similarities can be observed. Whilst EXPRESS data models have been analysed in greatest detail, due to the *force majore* coming from the IFC advancements and the practical studies performed on available EXPRESS-based schemas, a primary objective has been to develop a general specification encompassing a broader spectrum of object-oriented models. Thus, the style of CSML is actually more similar to SQL than to EXPRESS, because the goals of SQL are also more close to the goals of CSML than to EXPRESS.

5) *Support for the mapping development process* has not been a primary objective of the work. This is an issue related to the construction of a specifically implemented modelling framework for CEE[*)], and not to the representational capabilities of the mapping language, neither to the run-time functionality of the CEE system. However, to enable the data transformations between different, only partially harmonised model schemas, which is of great importance for the overall interoperability approach proposed in this thesis, the development of support utilities for the creation, documentation and maintenance of valid mappings with CSML had to be considered as well. For this purpose, two light-weight tools have been prototyped:

   – *Mapping Browser*[#)], which enables the graphical visualisation and inspection of a mapping specification by means of a graph structure of the inter-related classes of the source and target model schemas, with connecting "mapping nodes" corresponding to the MAP CLASS and COPY CLASS constructs of CSML;
   – *XML Converter*[##)], which enables the automated generation of hypertext documentation of mappings defined with CSML.

[*)] Such a framework should include all needed inter-model mapping specifications, along with the data model schemas themselves.

[#)] Developed as part of the KEE/LISP prototype environment.

[##)] Developed as a stand-alone tool in Java.

The Mapping Browser can be used to provide a quick overview of the principal interrelationships of the classes involved in a mapping; it allows also to examine interactively each “mapping node” w.r.t. the details of the encompassed attribute mapping patterns. The XML Converter can be used to provide a text-oriented presentation of a mapping which can then be viewed with a WWW-Browser; due to the included hypertext features, it can be especially helpful when specific constructs need to be traced down to the EXPRESS definitions of the respective source/target entity classes.
An impression of the Mapping Browser is provided on fig. 8.9 in chapter 8. The developed *XML DTD for CSML externalisation* is presented in appendix IV at the end of the thesis.

Aspects that need further work include:

1) *Consolidation of syntax*

When submitted to practical tests, CSML showed some typical teething troubles – a few constructs were not convenient in all cases, the use of NEW was not always obvious, often there were several different ways to express the same logical relationship, and the use of suffix operators did not justify itself. However, as all these are not serious defects, it should not be difficult to develop an improved version of the language.

2) *Enhancement of mapping templates*

Currently, some of the analysed mapping patterns require the use of complicated constructions. It would be desirable to have a more explicit presentation for all basic mapping cases.

3) *Generic function definition*

CSML itself provides a platform-independent notation but external functions can be specified only in LISP and, with some restrictions, in C. Although there is no known notation that provides for fully generic function definitions, it is worth considering to enhance the syntax and the respective mapping engine implementation so that standardised function interfaces, such as CORBA IDL (cf. OMG 1998), can also be applied.

4) *Method invocation*

Whilst currently not in the scope of CSML, in many cases the use of object methods would facilitate the definition of mapping conditions, such as instance selection, the access to complex data structures etc. An adaptation of the syntax in that respect should not be difficult to achieve, e.g. by extending the use of the addressing operator `'->'`. However, this must also be coupled with appropriate handling of arguments and return values, as well as with the use of the latter in a similar way as other CSML variables.

5) *Unit conversion*

Though unit conversion itself is a difficult problem when units are not explicitly supported in the involved models, it should be possible to provide an additional `UNITS` section in the class/attribute declarations to enable at least static and/or interactive specification of the necessary data. As simple as it seems, such feature is in fact of utmost importance in engineering environments.

6) *Pure object-oriented implementation*

   In a few cases the use of knowledge-based expressions was difficult to avoid because a replacement would have led to very complicated functions – see e.g. pp. 187, 194. However, as a pure object-oriented environment for CSML might also be required, it would be desirable to have more straight-forward alternative definitions for such cases. It is worth considering to introduce a small set of built-in functions for the tackling of typical condition specifications, as well as functional operators that could be used with or instead of the `APPLY` construct.

7) *Interactive mapping specifications*

   Mappings only of selected objects can be quite useful in interactive environments. Whilst in a **`map(…)`** operation a specific subset of entity classes that only need to be mapped may be provided, the mapping declarations themselves cannot be changed. It would be useful to be able to specify such mappings "on the fly", and include them directly in a mapping request, for example for modifying certain mapping conditions. This would lead to a feature, similar to *embedded SQL* with respect to CSML, which is worth considering for future extensions of the mapping system.

8) *User Interface*

   A graphical user interface cannot replace the hours of work needed to identify all needed equivalences between two schemas, but it can considerably facilitate the mapping development process and can help eliminate technical errors. However, the prototyped environment currently provides only limited browsing facilities to examine a mapping specification. A full software solution should include appropriate visual development aids which should allow not only the definition of mapping equivalences in graph-oriented object diagrams (like the ones provided for OM and VML), but could also include visualisation and navigation through the real-world building objects in a project model. The latter can be done with the integration of a geometry presentation tool, e.g. a VR-Browser, into the mapping system.
   An extension of the XML-based externalisation of CSML to provide better support for the data types involved in a mapping, e.g. by using the *XML Schema* specification proposed recently by the W3C (cf. Fallside et al. 2000), is also worth considering. In that respect, coordination with similar activities within the IAI and in STEP (cf. ISO 10303-28 1999) can be very fruitful.

Chapter 7: Mapping Case Studies

By three methods we may learn wisdom: First, by reflection, which is noblest; second, by imitation, which is easiest; and third by experience, which is the bitterest.

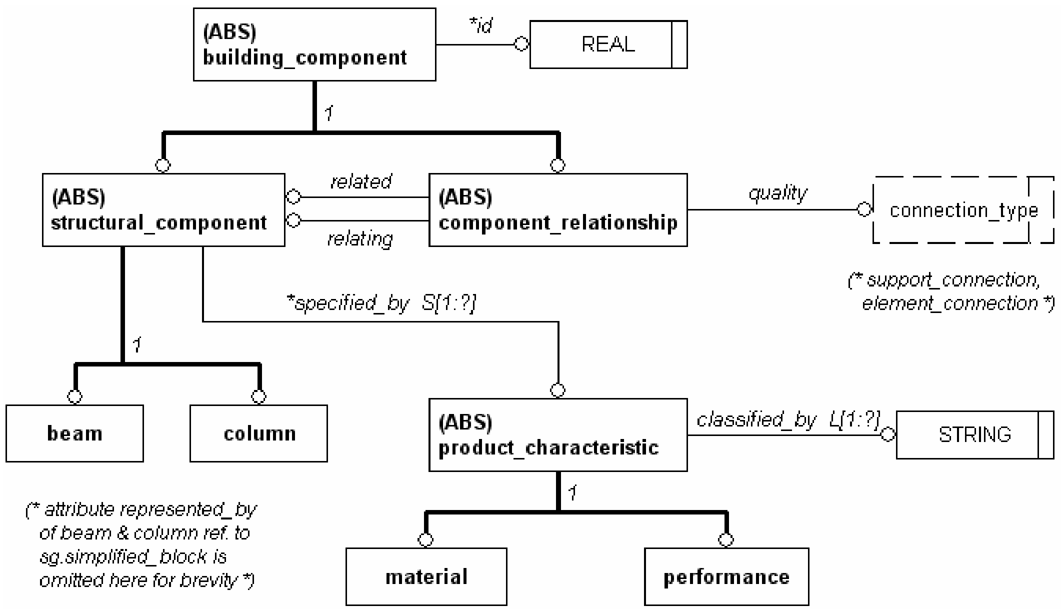
– Confucius

To gather evidence of the actual applicability of CSML, several practical tasks have been examined. This chapter, completing the presentation of the language, discusses three case studies of more comprehensive mapping problems performed in the course of the work. Compared to the illustrative examples given in the previous chapter, the scope of these studies is broader, and the involved schemas are more complex, closer to real practical cases. For better overall view, the source and the target schemas for each of the discussed test cases are presented in EXPRESS-G. All lines in the mapping specifications are numbered to help referencing them in the accompanying comments.

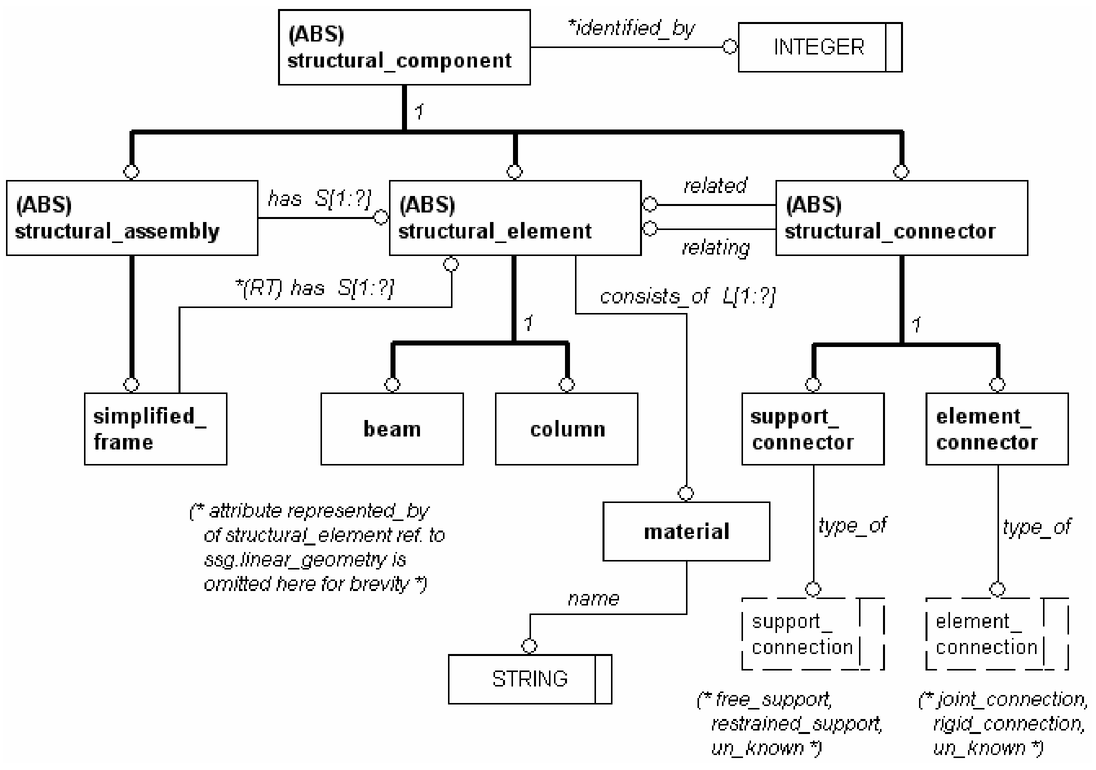
7.1 Test Bed Model Transformation Example

This test example presents a slightly modified version of a study which has been conducted by Liebich et al. (1995) to survey different mapping methods available on the product modelling arena. The original problem comprises several mapping exercises between two pairs of EXPRESS schemas, a building system structural component schema (bssc) and a simple geometry schema (sg) as source schemas, along with a preliminary structural system schema (pss) and a structural system geometry schema (ssg) as target schemas. Here the two complementary geometry schemas (sg and sgg) are omitted for brevity. The bssc and the pss schemas shown below are respectively severed from all references to sg and sgg.

***Source schema* (bssc):**


(ABS) building_component
*id
REAL
1
(ABS) structural_component
related
relating
(ABS) component_relationship
quality
connection_type
(* support_connection, element_connection *)
*specified_by S[1:?]
1
beam
column
(ABS) product_characteristic
classified_by L[1:?]
STRING
1
(* attribute represented_by of beam & column ref. to sg.simplified_block is omitted here for brevity *)
material
performance

Target schema **(`pss`):**


(ABS) structural_component
*identified_by
INTEGER
1
(ABS) structural_assembly
has S[1:?]
(ABS) structural_element
related
relating
(ABS) structural_connector
*(RT) has S[1:?]
consists_of L[1:?]
1
1
simplified_ frame
beam
column
support_ connector
element_ connector
(* attribute represented_by of structural_element ref. to ssg.linear_geometry is omitted here for brevity *)
material
type_of
type_of
name
support_ connection
element_ connection
STRING
(* free_support, restrained_support, un_known *)
(* joint_connection, rigid_connection, un_known *)

Liebich et al. consider the following five mapping problems most essential in their study:

[1] **`bssc:material` → `pss:material`.**
“Attribute to entity” mapping with 1:N cardinality, depending on the actual cardinality of the attribute `bssc:material.classified_by`.

[2] **`bssc:column` → `pss:column`** and in analogy **`bssc:beam` → `pss:beam`.**
Mainly 1:1 “attribute to attribute” conditional mapping, depending on the entity name, and involving type conversion for the mapping `id` → `identified_by`.

[3] **`bssc:component_relationship` → `pss:support_connector`** and **`bssc:component_relationship` → `pss:element_connector`.**
Conditional mappings depending on the value of the `quality` attribute of `bssc:component_relationship` and reusing the results of [2].

[4] **`bssc:structural_component` → `pss:simplified_frame`.**
Conditional creation of new `pss:simplified_frame` instances depending on the relationships between the instances of `bssc:component_relationship`, `bssc:column` and `bssc:beam` (this is not considered a typical type of a mapping problem, but a complex precondition for a mapping).

[5] **`sg:simplified_block` → `ssg:linear_geometry`.**

The fifth of these problems is not shown here for brevity. It requires complex geometric transformations that would have made the detailed mapping specification considerably longer, adding only little value to the presentation. Such geometric transformations are discussed in detail in the case study presented in section 7.3. All other formulated tasks are covered below.

Mapping specification:

```
     (MAP pss FROM bssc
      CLASSES
       (MAP CLASS structural_component FROM building_component
         ATTRIBUTES
           (MAKE identified_by FROM id MAP INTEGER (LAMBDA (X) X) )
         )
[2]    (MAP CLASS structural_element FROM structural_component
         ATTRIBUTES
           (MAKE consists_of FROM (ASSOC material REF specified_by))
         )
       (COPY CLASS column FROM column)
       (COPY CLASS beam FROM beam)
       (MAP CLASS structural_connector FROM component_relationship
         ATTRIBUTES (SAME related relating)
         )
       (MAP CLASS support_connector FROM component_relationship
         CONDITIONS (quality = support_connection)
         ATTRIBUTES
           (MAKE type_of FROM
             (USER-CHOICE ONEOF
                (LISTOF FREE_SUPPORT RESTRAINED_SUPPORT UN_KNOWN)
[3]             "Choose support type"))
         )
       (MAP CLASS element_connector FROM component_relationship
         CONDITIONS (quality = element_connection)
         ATTRIBUTES
           (MAKE type_of FROM
              (USER-CHOICE ONEOF
                 (LISTOF JOINT_CONNECTION RIGID_CONNECTION UN_KNOWN)
                 "Choose support type"))
         )
       (MAP CLASS simplified_frame FROM structural_component
         VAR (MAKE ?ELEMENT_LIST FROM
               (USER-CHOICE LISTOF
                  (DESCENDANTS structural_element)
                  "Choose frame elements"))
         CONDITIONS (?ELEMENT_LIST <> NUL)
[4]      EXCLUSIONS (structural_component FOR identified_by)
         ATTRIBUTES
           (MAKE has FROM ?ELEMENT_LIST)
           (MAKE identified_by
              CONSTRUCTOR make_assembly_identifier
                          ARGS ?ELEMENT_LIST -> identified_by )
         )
      DEPENDENT-CLASSES
       (MAP CLASS material FROM material
[1]      ATTRIBUTES (MAKE name FROM (ITERATE-ON classified_by))
         )
      PRESETS (LOAD "fn-file.lisp" FOR make_assembly_identifier)
     )
```

Comments:

The addressed mapping problems are tackled with CSML as follows[*)]:

[1] The mapping **`bssc:material` → `pss:material`** is defined in lines 45 to 48.
This is a typical "dependent-class" in the terminology of CSML as it is not meaningful to create instances of `pss:material` that are not referenced by any structural component in the `pss` model.
From the EXPRESS-G diagrams and the explanation given in (Liebich et al. 1995) it is not absolutely clear what mapping type exactly is needed here (by the way, a common problem in the development of mapping specifications). However, after examining the full EXPRESS listing of the models, it appears that an instance of `bssc:structural_component` may contain 0 or 1 references to an instance of `bssc:material` which, in turn, may contain a list of several materials identified merely by their names given in the attribute `classified_by`. In the `pss` model a separate `material` instance must be created for each of these names. These instances are then associated with the corresponding instance of `pss:structural_element` as illustrated in the sample instance-level mapping graphed below.

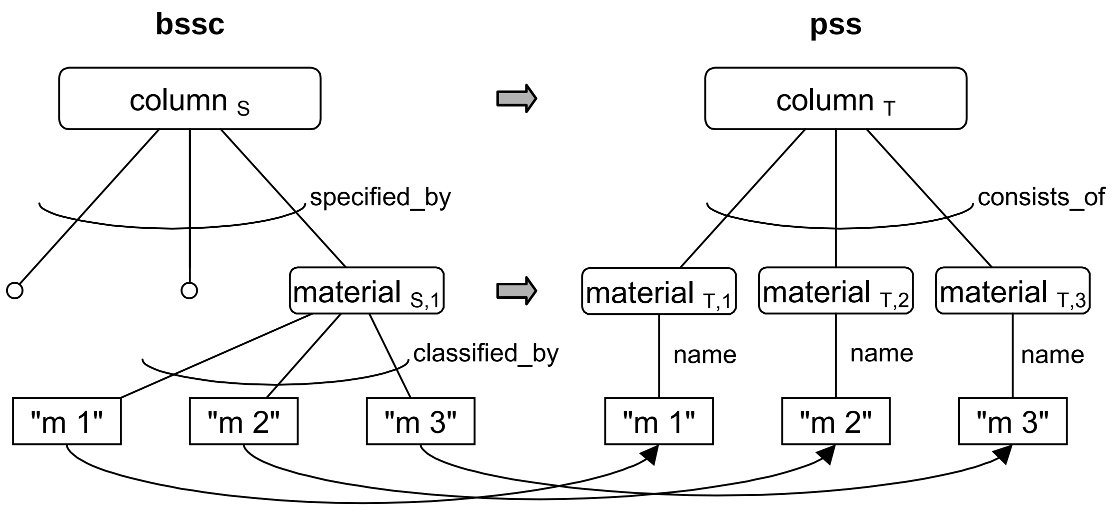


Fig. 7.1: Instance level mapping of entity 'column' and the associated entity 'material'
(three attributes in bssc:material$_{S,1}$ lead to the generation of three entities pss:material, with 'name' attribute corresponding to the respective element in the list 'classified_by')

This 1:N mapping problem is tackled in line 47 of the mapping specification with the help of the template operator ITERATE-ON which initiates an iteration over all elements contained in the attribute `classified_by`, creating one instance of `pss:material` for each element, and then assigning the value of this element to the attribute `name` of the created new instance of `pss:material`. Without ITERATE-ON there would be a 1:1 correspondence between `bssc:material` and `pss:material` which would not be correct.

*) The numbers in brackets correspond to the respectively labelled blocks in the mapping specification code.

[2] The mappings for **`pss:column`** and **`pss:beam`** are covered in lines 3 to 12.
First, the mapping of the top-level abstract superclass `bssc:building_component` is defined. It involves only the 1:1 mapping `id` → `identified_by` with an associated, somewhat artificial type conversion. This attribute mapping is specified on line 5 with a common, somewhat cumbersome idiom provided in CSML for such purposes. The applied LAMBDA function merely returns its argument, converting it to the required INTEGER data type.*)
The mapping of the abstract entity `bssc:structural_component` reuses automatically the mapping for `bssc:building_component` by inheritance, and defines in addition the transformation `specified_by` → `consists_of` with the help of the ASSOC operator (line 9). This allows to associate all new instances of `pss:material`, created by the mapping `bssc:material` → `pss:material` and referenced in the attribute `specified_by` of `bssc:structural_component`, with the attribute `pss:structural_element.consists_of`.
The mapping declarations for `bssc:column` and `bssc:beam` are themselves quite trivial as they can be reduced merely to reusing the mapping declarations for their superclasses by inheritance (lines 11, 12) **).

[3] Lines 13-31 detail the mappings needed for **`pss:structural_connector`**, **`pss:support_connector`** and **`pss:element_connector`**.
Here again, a mapping for the superclass (`bssc:component_relationship` → `pss:structural_connector`) is defined first, so that it can be reused in the subclasses (lines 13-15).
The mappings for `pss:support_connector` and `pss:element_connector` are identical in structure. Both use the interactive template operator USER-CHOICE enabling run-time assignment of values for the attribute `type_of` (lines 19-22 and 27-30 respectively), along with the implicit inheritance of the mapping declaration for `pss:structural_element`.

[4] At last, lines 32-44 present the mapping for **`pss:simplified_frame`**.
The intent of the original study here has been to explore the existing associations in the source model, i.e. to find for each instance $\texttt{c} \in \{\ \texttt{component\_relationship}\ \}$

$\texttt{c.relating} = \uparrow s_i \wedge \texttt{c.related} = \uparrow s_j$

where $s_i, s_j \in \{\ \texttt{structural\_component}\ \}$

and use these as a complex precondition to form subsets of `structural_component` for the aggregation relationship defined by `pss:simplified_frame.has`, tracking the topology of the components as appropriate.
There is no convenient way to express such complex preconditions in CSML, other than by an external function. On the other hand, I don't believe it to be the correct approach in this particular case anyway, because the decision about the definition of

*) Strictly speaking, this LAMBDA function is actually superfluous here because simple type conversions can be tackled automatically by the prototyped mapping engine on the basis of the available entity definitions in the model schemas. However, in general such conversions have to be specified explicitly. Line 5 demonstrates the principal procedure.

**) Of course, in a real-life situation `beam` and `column` may contain several other attributes that are not provided in this case study.

frames in a structural system – an essential part of a structural engineer's work in preliminary design – requires deeper knowledge than the simple element connectivity expressed by the attributes *relating/related* can represent.

Therefore, for this mapping a completely different approach is taken. At first, the user is given the opportunity to interactively select the elements of a frame out of all generated `structural_element` entities in the `pss` model according to his own knowledge and judgement (lines 33-36). Admittedly, this is also not very likely to be of much practical use because the information provided to the user by the mapping alone would be minimal and in an inconvenient presentation format. However, it can be enhanced by a dedicated client function e.g. by combining it with a geometric view of the structure, highlighting the questioned structural elements and allowing to select them visually. Alternatively, the user may simply choose to ignore the request which will leave the variable `?ELEMENT_LIST` unset and, because of the condition on line 37, no `simplified_frame` entities will be generated, but there will be also *no wrong assumptions made automatically*. In this case the mapping will remain incomplete, but it can be complemented by an intelligent *structural design assistant*, taking on the task of a mapper as proposed in (Hauser et al. 1996).

The EXCLUSIONS condition given on line 38 prevents from performing the mapping for `simplified_frame.identified_by` twice – by inheritance, from the mapping for `structural_component` (line 5), and according to the specification given on lines 41 to 43, overriding the former result.

Since there is no equivalent ID that can be adopted from the `bssc` model for a `simplified_frame` instance in `pss`, an external function, not further detailed here, is used to create the values for the attribute `simplified_frame.identified_by`.

Finally, using the list of structural elements stored in `?ELEMENT_LIST`, the values for `simplified_frame.has` can be obtained by simple 1:1 equivalence (line 40).

As a whole, because of the interdependencies of the mapping declarations, the `pss` model will be populated according to the following **iteration sequence** (not taking into account intermediate iteration steps generating incomplete, "skeleton" instances):

1) Generating the instances of **`material`**;
2) Generating the instances of **`beam`** and **`column`** – depending on `material`;
3) Generating the instances of **`simplified_frame`** – depending on `beam` and `column`.
4) Generating the instances of **`support_connector`** and **`element_connector`** – depending on `beam` and `column`.
5) Pruning the instances of `material` that are not associated to any instances of the subclasses of `structural_element` in `pss`.

Considering only the pure interdependencies between the source and target instances involved in the mapping, `support_connector` and `element_connector` could be generated actually in the same step as `simplified_frame`. However, in the implementation of the prototyped mapping engine all mappings requiring user input are processed *before* the mappings that can be resolved automatically. Since a mapping operation can take a long time in practical cases (several minutes or even hours), this policy gives the user the opportunity to react as early as possible to all associated requests.

7.2 Design Interaction Example

This case study presents a moderately sized practical example drawn from an early rapid-prototype implementation of CSML in the COMBI project (Katranuschkov & Scherer 1995). It depicts the forward and inverse mapping specifications needed to transform typical structural objects defined in a structural domain model, such as *nodes* and *nodal results*, into objects typical for a foundation design application, and vice versa. Such transformations are necessary to enable the collaborative design interaction cycle *preliminary structural design* ↔ *foundation design* to be performed properly and with as little as possible information and time loss. The presented mapping task involves basically the transformation of reactions into loads and, inversely, computed foundation stiffness into restraints for the elastic supports of columns, frames and shear walls as shown schematically on fig. 7.2 below. It is a typical example for the often very different representations of building objects a mapping system has to deal with.

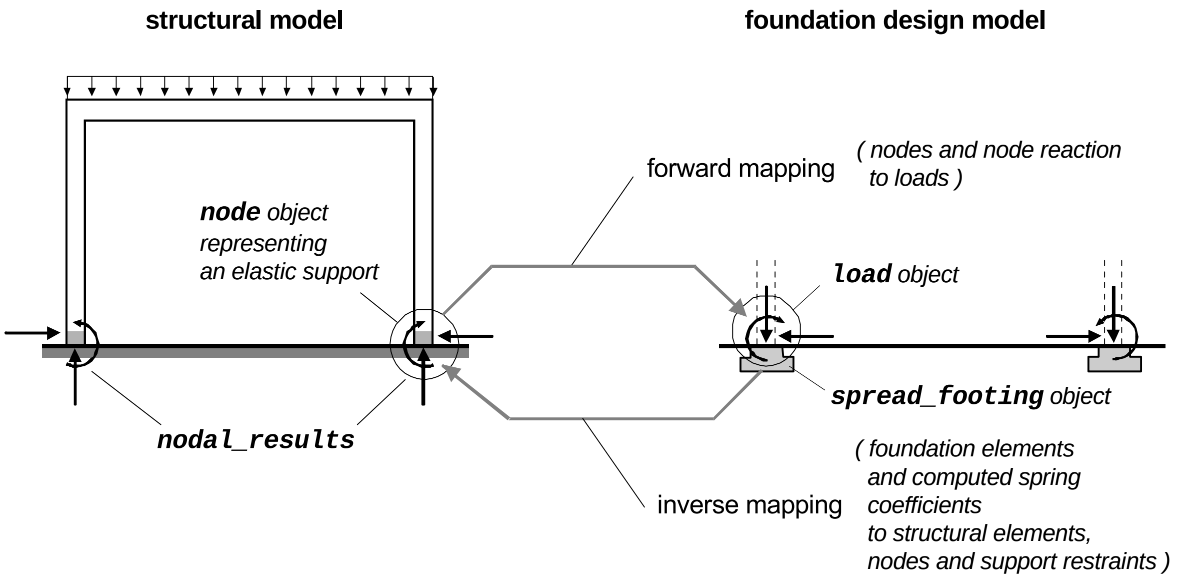

structural model
foundation design model
forward mapping
(nodes and node reaction to loads)
node object representing an elastic support
load object
spread_footing object
nodal_results
(foundation elements and computed spring coefficients to structural elements, nodes and support restraints)
inverse mapping

Fig. 7.2: Schematic view of the mapping transformations needed for the design cycle preliminary structural design – foundation design

For compactness, both the source and the target schemas are considerably reduced here compared to the original problem.

In the structural domain model (`struct_domain_model`) only entities directly involved in the mapping are shown; all subclasses of `structural_element` are severed as they do not contain data of specific interest for the mapping, and geometric characteristics are reduced to the simple link `node.location` = `↑p | p ∈ { point }`.

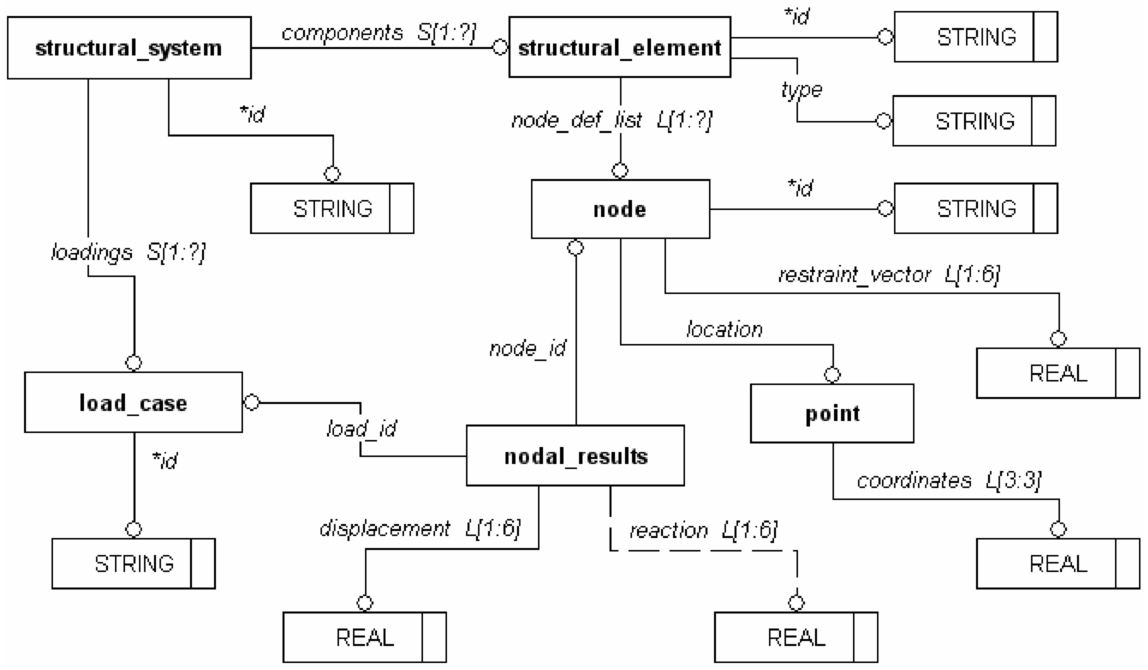
In the application-specific foundation design model (`fd_appl_model`) all properties of subclasses of `foundation_element` are left out, and so are all entities representing the geotechnical characteristics of the site; the remaining part of the model is reproduced from the COMBI project without further modifications.

In both schemas *measure units* are substituted with *real* values, assuming that the same units are used in both model representations.*)

As the main focus of this test case has been to show how *design interaction* can be enhanced by enabling simultaneous work on different aspects of a gradually filled evolving structural model to achieve a better thought out common solution, both the forward and the inverse mapping will be demonstrated.

These two mapping specifications are both ***partial***, as it is not meaningful to populate completely a `fd_appl_model` from the data contained in a `struct_domain_model` – a task dedicated to the foundation design application itself, neither possible to fully reconstruct a `struct_domain_model` only by the results of a foundation design application.

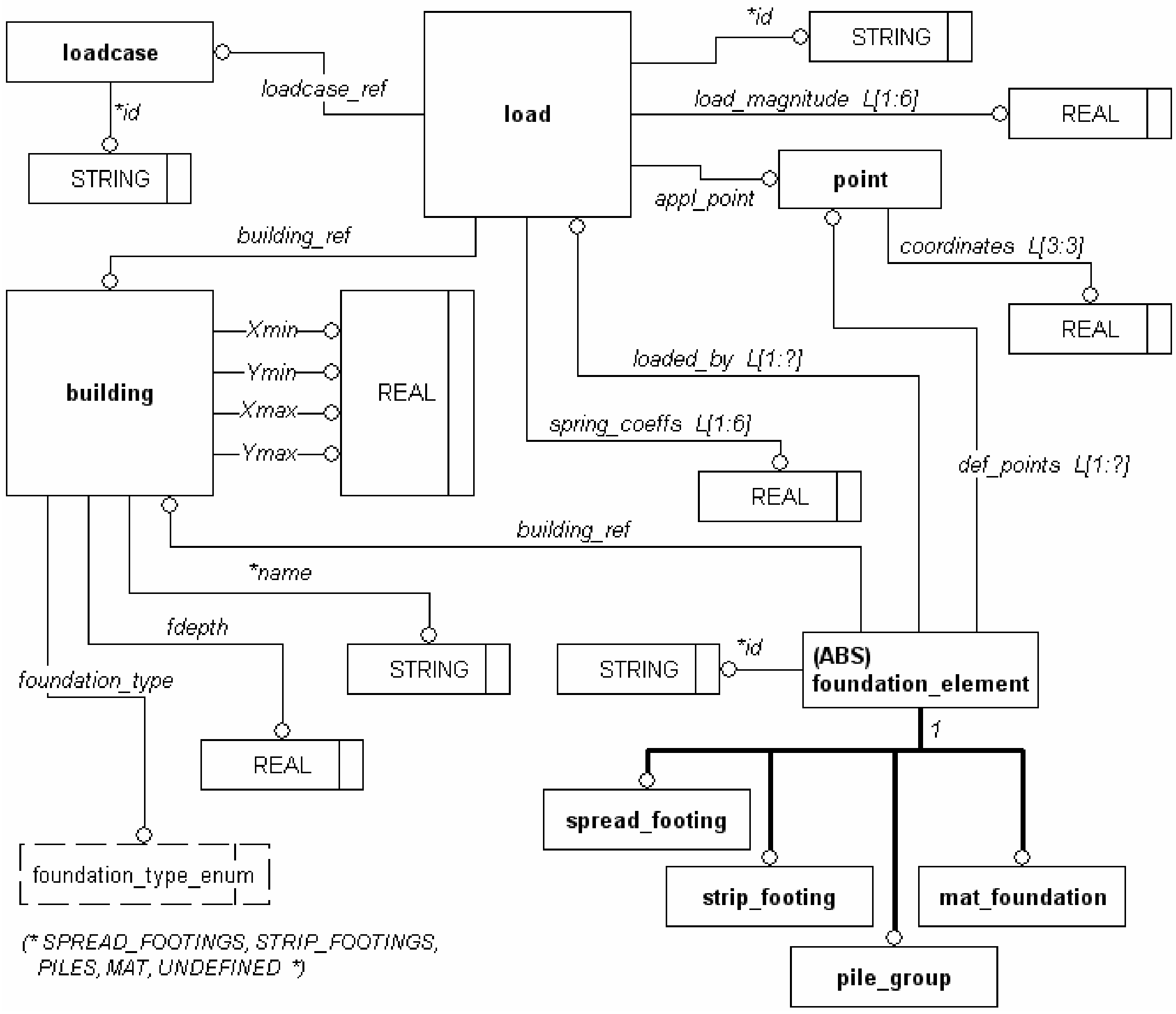
Source schema **(part of "`struct_domain_model`") :**



*) In the original structural domain model of the COMBI project the reference geometry of the structural elements is more complicated, with great similarity to the IFC approach. The full model contains a total of 67 entity classes, with additional 45 resource entities for the geometry, and 17 entities for the material characteristics of the primary structural objects. The full foundation design application model contains a total of 14 entity classes with a few additional resource entities representing geometric, material and soil properties (Ammermann et al. 1994).

The design tools used in the original COMBI example have been developed at the Dresden University of Technology (Knowledge-based Assistant for Preliminary Structural Design, PRED), and at D'Appolonia, Genoa (Foundation Design Expert System, PAULA). An overview of these tools is provided in (Scherer & Sparacello 1996).

Target schema **(part of "`fd_appl_model`") :**



The **forward mapping specification** detailed on the next page encompasses the following mapping exercises:

[1] **Declaration of global variables** to be used in more than one inter-class mapping specification;

[2] **`struct_domain_model:structural_system → fd_appl_model:building.`** Creation of two attributes not contained in the source model along with a functional transformation for the attributes `xmax`, `xmin`, `ymax`, `ymin` of `building`.

[3] **`struct_domain_model:load_case → fd_appl_model:loadcase`** and **`struct_domain_model:nodal_results → fd_appl_model:load.`** Central issue of the mapping specification including a simple synonym problem for `loadcase`, and a sophisticated transformation for `load` with two filtering conditions, several 1:1 mappings of different complexity, and a mapping with a post-condition for `building_ref`, creating new links in the target model.

[4] **Implementation of two external functions** needed for the involved functional mapping patterns.

Forward mapping:

```
     (MAP PARTIALLY fd_appl_model FROM SHARED struct_domain_model
      PRESETS
       (MAKE ?eps FROM 1.0E-6)
[1]    (MAKE ?elevation FROM
          (USER-INPUT REAL "Found. elevation [m] <0.0>:"
             DEFAULT 0.0) )
      CLASSES
       (MAP CLASS building FROM structural_system
         ATTRIBUTES
           (SAME name AS id)
           (MAKE fdepth FROM
              (USER-INPUT REAL "Found. depth <0.0>:" DEFAULT 0.0)
              ?elevation
              APPLY (LAMBDA (X Y) (- Y X)) )
           (MAKE foundation_type FROM
[2]           (USER-CHOICE ONEOF (LISTOF SPREAD_FOOTINGS
                                         STRIP_FOOTINGS
                                         PILES MAT UNDEFINED)
                 "Choose found.type" DEFAULT UNDEFINED) )
           (MAKE Xmin Xmax Ymin Ymax
              CONSTRUCTOR safd-get-foundation-coord
                          ARGS (DESCENDANTS node) ?elevation ?eps)
         )
       (COPY CLASS loadcase FROM load_case)
       (MAP CLASS load FROM nodal_results
         CONDITIONS
           (PRED (THIS HAS AT LEAST 1 reaction))
           (PRED safd-check-foundation-point
                 ARGS (REF node_id FOR location) ?elevation ?eps)
         ATTRIBUTES
           (MAKE id FROM (REF node_id FOR id) )
           (MAKE building_ref FROM
[3]           (NEW building
                 WHEN (load_id =
                       (ONEOF structural_system -> loadings))))
           (MAKE loadcase_ref FROM load_id)
           (MAKE load_magnitude FROM reaction
                                MAPCAR (LAMBDA (X) (- X)) )
           (MAKE appl_point FROM (REF node_id FOR location) )
           (MAKE spring_coeffs FROM
              (REF node_id FOR restraint_vector))
         )
      DEPENDENT-CLASSES
       (COPY CLASS point)
      PRESETS
       (LOAD "sa-fd-map.lisp"
[4]          FOR safd-get-foundation-coord
                 safd-check-foundation-point
         )
      )
```

The function file **sa-fd-map.lisp**, in which the external Common LISP functions are defined, has the following content:

```
;;; -*- Syntax: Common-Lisp; Base: 10; Mode:LISP; Package: KEE -*-
;;;
;;; Author:   P. Katranuschkov
;;; Synopsis: Defines the external functions that are needed for
;;;           mapping struct_domain_model to fd_appl_model.
;;;           In each function the arguments FromClass & NewClass
;;;           are provided automatically by the Mapping Engine.
;;; Uses:     get.value(Unit Slot) – the generic KEE function for
;;;                                  accessing object attributes

(DEFUN safd-check-foundation-point (FromClass NewClass Pt elev eps)
  (declare (ignore FromClass NewClass))
  (< (abs (- (third (get.value Pt 'COORDINATES)) elev)) eps)
)

(DEFUN safd-get-foundation-coord (FromClass NewClass
                                  NodeList elev eps)
  ;; initialisation of X/Y max/min
  (let ((Xmax -1.0E99) (Ymax -1.0E99)
        (Xmin  1.0E99) (Ymin  1.0E99))
    ;; the following expression selects only nodes that lie on
    ;; the foundation and stores them in 'NodeList' modifying
    ;; its initial content.
    (setf NodeList
       (remove-if-not
          #'(lambda (u)
              (safd-check-foundation-point FromClass NewClass
                                           (get.value u 'LOCATION)
                                           elev eps))
          NodeList))
    ;; loop through the list of all found elements
    ;; to check their coordinates and determine
    ;; the dimensions of the building foundation
    (loop for Obj in NodeList
          for Pt = (get.value Obj 'LOCATION)
          do
          (when (get.value Obj 'SUPPORT)
            (let ((xt (first  (get.value Pt 'COORDINATES)))
                  (yt (second (get.value Pt 'COORDINATES))))
              (when (< xt Xmin) (setf Xmin xt))
              (when (< yt Ymin) (setf Ymin yt))
              (when (> xt Xmax) (setf Xmax xt))
              (when (> yt Ymax) (setf Ymax yt)))))
    ;; return a list of the coordinates of the found. boundary
    (values Xmin Xmax Ymin Ymax)
  )
)
```

Comments:

In the above mapping specification the identified problems are tackled in the following way:

[1] Lines 3-6 show the value assignments for the global variables **`?eps`** and **`?elevation`**, the first being a simple base type constant assignment (line 3), and the second requiring user response, along with a foreseen default value (lines 4-6).
The foundation elevation (`?elevation`), not available in the source model, is needed for checking which nodes do actually lie on the foundation plane and must therefore be provided interactively for the whole mapping to succeed; the default value of 0.0 is included in the prompt string as a hint to the user that a value for `?elevation` will be assumed in any case.
The variable `?eps` defines explicitly the numeric precision for floating point computations.
Both variables are defined globally as they are needed in more than one class mapping declarations (see lines 13, 22, 29).

[2] Lines 8-23 cover the mapping for **`fd_appl_model:building`**.
It involves a simple 1:1 equivalence for the attribute `name` (line 10), the generation of two new attributes, `fdepth` and `foundation_type`, with the help of the interactive templates USER-INPUT and USER-CHOICE (lines 11-14 and 15-19 respectively), and a complex value assignment for the attributes `xmin`, `xmax`, `ymin` and `ymax`, depending on the result returned by an applied external function (lines 20-22).
The attributes `fdepth` and `foundation_type` are missing in the source model and *must* be provided interactively; otherwise the mapping cannot be performed without error. The USER-INPUT template applied for `fdepth` expects a real value as response, assuming a default value of 0.0 when no response is given. The USER-CHOICE template applied for `foundation_type` contains as selection list the full set of enumeration items defined for `foundation_type_enum` in schema `fd_appl_model`; it allows only one of these items to be chosen in response, due to the parameter ONEOF on line 15 (assuming the default value UNDEFINED when no response is provided).
`xmin`, `xmax`, `ymin` and `ymax` represent the enclosing rectangle of the building's contour at ground level. Their values cannot be deduced directly from the information contained in a `structural_system` instance, but they can be determined procedurally, by scanning the coordinates of all nodes defined in the source model, filtering out the nodes that do not lie on the foundation plane, and then calculating the corner points of the enclosing rectangle. For this purpose, the external function `safd-get-foundation-coord` is used. This function returns four values as a list, which is then *destructured* to obtain the values of the four attributes `xmin`, `xmax`, `ymin` and `ymax` respectively.

[3] The mappings for **`fd_appl_model:loadcase`** and **`fd_appl_model:load`** are detailed on lines 24 to 42.
The instances of `loadcase` in the target model are obtained simply by copying the data representing the instances of `load_case` in the source model. This is possible because the only difference between `struct_domain_model:load_case` and `fd_appl_model:loadcase` is in the class names (line 24).

The instances of `load` in the target model are obtained from a subset of the instances of `nodal_results` in the source model. This subset is determined by applying two pre-conditions to the mapping: a knowledge-based template selecting only those instances that contain *node reactions* (line 27), and an external predicate function checking if the corresponding `node` instance referenced through `nodal_results.node_id` lies on the foundation plane (lines 28-29).

The attribute values of the instances of `load` are then generated with the help of 1:1 attribute mappings as follows:

– `load.id` (line 31) is determined by using the REF operator to obtain the “tele-scoped” value of `node.id` referenced through `nodal_result.node_id`, i.e.

  $\forall$ `x` $\in$ `{ nodal_results } ( x.node_id =` $\uparrow$`n | n` $\in$ `{ node } )` $\Rightarrow$
  `n.id` $\rightarrow$ `l.id | l` $\in$ `{ load } ;`

– `building_ref` (lines 32-35) is determined by using the NEW operator with a post-condition (WHEN) to select from any new instances of `building` the one for which the *load case* referenced by the current instance, `x`, of `nodal_results` is also referenced by the instance `s` of `structural_system` mapped to that particular `building` instance – a typical case of a conditional “inverse transitive” pattern, i.e.:

  $\forall$ `x` $\in$ `{ nodal_results } ( x` $\rightarrow$ `ld )` $\Leftrightarrow$
  $\exists$`! b : structural_system(s)` $\rightarrow$ `building(b)` $\wedge$
  `( x.load_id =` $\uparrow$`lc | lc` $\in$ `{ s.loadings } )`
  where `ld` $\in$ `{ load }, lc` $\in$ `{ load_case } ;`

– `loadcase_ref` is obtained by a simple equivalence specification with respect to `nodal_results.load_id` (line 36);

– `load_magnitude`, which should contain a list of 6 values, is obtained through a functional set equivalence w.r.t. the attribute `nodal_results.reaction` which is defined as a list of the same length; the directions of the node reactions are inverted by multiplying them by `–1.0` with the help of the suffix operator MAPCAR (lines 37-38);

– `appl_point` and `spring_coeffs` are obtained in a similar way as `id`, using the REF operator to retrieve the pointer to a `point` instance (line 39), and the values of `n.restraint_vector | n` $\in$ `{ node }` for `nodal_results.node_id =` $\uparrow$`n` respectively (lines 40-41).

[4] The Common LISP functions used in the more complicated functional transformations requiring procedural processing (lines 21, 28) are specified in an external file, *sa-fd-map.lisp*, and are loaded prior to the execution of the mapping with the help of the LOAD construct (lines 46-49).

Such functions, complementing the declarative style of CSML, are generally not difficult to write because they have to solve only partial, dedicated mapping subtasks with pronounced algorithmic character. However, they are inevitably more tightly linked to the representation paradigm and the implementation platform of the mapping engine and are therefore, unlike other constructs of CSML, not completely platform-independent.

To give an impression, the full content of the file *sa-fd-map.lisp* was listed on page 213.

As a whole, the interdependencies of the declarations in the forward mapping lead to the following **iteration sequence** of populating the target `fd_appl_model`:

1) Assigning the values of all **global variables**;
2) Generating the instances of **building**, depending on the values of ?elevation and ?eps, and the instances of **loadcase**;
3) Generating the instances of **load**, depending on building and loadcase, and the associated instances of **point**.

The required **inverse mapping** is shorter. It comprises the following mapping tasks:

[5] **fd_appl_model:building → struct_domain_model:structural_system.**
Incomplete mapping including an "inverse association" (N:1 → 1:N) for the attribute components of structural_system, related to the extent of another mapping, [7], reduced by a filtering condition.

[6] **fd_appl_model:load → struct_domain_model:node**.
Re-creation of node entities with different types of 1:1 mappings (the data contained in load instances are actually not expected to be changed by the foundation design application, but they are needed in the new structural domain model for the proper re-construction of the relationships between other entity instances).

[7] **fd_appl_model:foundation_element → struct_domain_model:structural_element**.
A 1:1 mapping using an external function for the attribute id, along with a complex 1:N mapping for node_def_list, depending on the mapping for node.

Inverse mapping:

```
    (MAP PARTIALLY SHARED struct_domain_model FROM fd_appl_model
     CLASSES
      (MAP CLASS structural_system FROM building
        ATTRIBUTES
          (MAKE id CONSTRUCTOR make-unique-id ARGS name)
[5]       (MAKE components FROM
             (ASSOC structural_element
                WHEN (structural_element :=: foundation_element)
                     (foundation_element -> building_ref =
                      building -> THIS)))
        )
      (MAP CLASS node FROM load
        ATTRIBUTES
          (MAKE id CONSTRUCTOR make-unique-id ARGS load -> id)
[6]       (SAME location AS appl_point)
          (SAME restraint_vector AS spring_coeffs)
        )
      (MAP CLASS structural_element FROM foundation_element
        ATTRIBUTES
          (MAKE id CONSTRUCTOR make-unique-id
[7]                    ARGS foundation_element -> id)
          (MAKE type FROM CLASSNAME)
          (MAKE node_def_list FROM (ASSOC node REF loaded_by))
        )
     DEPENDENT-CLASSES (COPY CLASS point)
     PRESETS (LOAD "make-unique-id.lisp" FOR make-unique-id)
    )
```

Comment:

The separate subtasks associated with this mapping are tackled as follows:

[5] Lines 3 to 11 detail the transformations that are necessary for **`structural_system`**. The 1:1 mapping for the attribute `id` (line 5) is performed with the help of an external function, `make-unique-id`, which computes its return value in accordance with the assumed convention for object identification in a shared repository, at the same time guaranteeing that the generated `id` is unique. If the target model had not been shared, this attribute mapping could have been replaced simply by `(SAME id AS name)`.
The mapping needed for the attribute `components` is more complicated. It depicts a typical situation where a N:1 relationship in the source model has to be "inverted" to a 1:N association in the target. In this particular case, the task is:

```
∀ b,s ( building(b) → structural_system(s) )
derive:  s.components = [ ↑$e_1$ ... ↑$e_n$ ]
where:   ∀ ↑$e_i$ ∈ [ ↑$e_1$ ... ↑$e_n$ ]
           ∃! f : foundation_element(f) → structural_element($e_i$) ∧
                  f.building_ref = ↑b .
```

For this task the ASSOC operator is applied to the instances of `structural_element`, with a post-condition enabling to select the subset corresponding to the instances of `foundation_element` for which the given predicate is satisfied (lines 6-10). The keyword THIS on line 10 instructs the mapping engine to use the pointers to the instances of `building` in the relational operator comprising the body of the predicate.

[6] The mapping transformations for **`struct_domain_model:node`** are easier to write. They involve two simple 1:1 correspondences specified in similar way (a list of entity references for `node.location` on line 15, and a list of real values for `node.restraint_vector` on line 16), along with a 1:1 mapping for `node.id` using the same functional transformation as for `structural_system.id` (line 14).

[7] The mapping for **`struct_domain_model:structural_element`** detailed on lines 18 to 24 comprises three different types of attribute declarations. The values for `id` are obtained in the same way as for `structural_system` and `node` (lines 20-21). The string values for `type` are taken from the class name of the source entity, i.e. "spread_footing", "strip_footing", "pile_group" or "mat_foundation" (line 22). At last, the values for `node_def_list`, which should be lists of references to `node`, are determined by the correspondence of the new instances of `node` to the instances of `load` referenced through `foundation_element.loaded_by` in the source model (line 23).

Here, the **iteration sequence** of populating the target model is:

1) Generating the instances of **`node`** and the associated instances of **`point`**;
2) Generating the instances of **`structural_element`** – depending on `node`;
3) Generating the instances of **`structural_system`** – depending on `node` and `structural_element`;
4) At last, if necessary, pruning of *points* not associated to *nodes.*

As it can be seen, the sequence of operations undertaken by the mapping engine does not necessarily follow the order of the mapping declarations in a CSML specification. This underpins the declarative character of the language.

Fig. 7.3 on the next page demonstrates graphically the described interaction procedure.

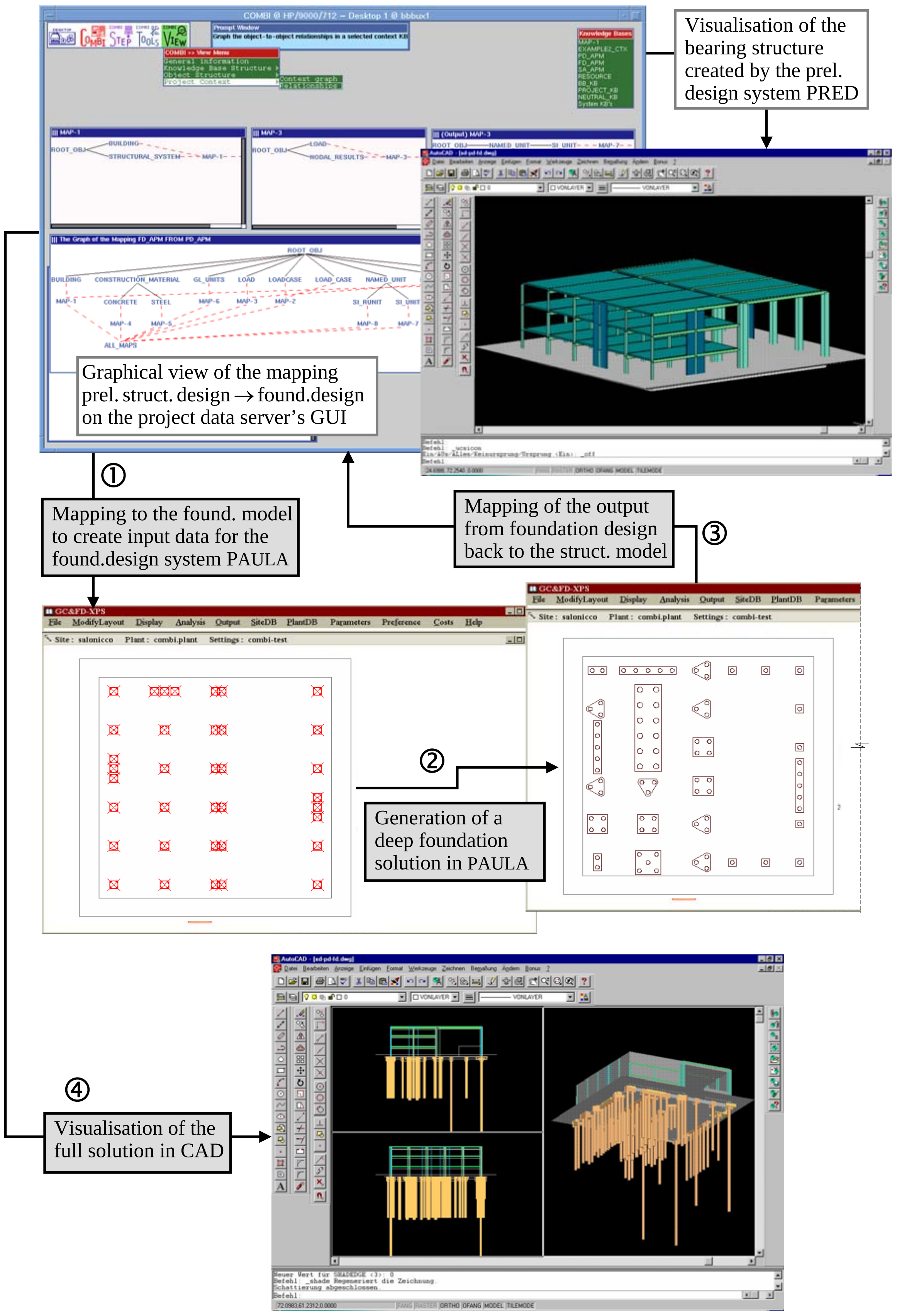

Visualisation of the bearing structure created by the prel. design system PRED
Graphical view of the mapping prel. struct. design → found.design on the project data server's GUI
①
Mapping to the found. model to create input data for the found.design system PAULA
Mapping of the output from foundation design back to the struct. model
③
②
Generation of a deep foundation solution in PAULA
④
Visualisation of the full solution in CAD
COMBI @ HP/9000/712 – Desktop 1 @ bbbux1
Knowledge Bases
The Graph of the Mapping FD_APM FROM PD_APM
ROOT_OBJ
GC&FD-XPS
File ModifyLayout Display Analysis Output SiteDB PlantDB Parameters Preference Costs Help
Site: salonicco Plant: combi.plant Settings: combi-test
AutoCAD - [ed-pd-fd.dwg]

Fig. 7.3: Graphical presentation of the interaction "structural design – foundation design"

7.3 VR View Generation for IFC-Based Product Data

In many situations a model transformation might be needed for some presentation purpose, e.g. to derive a VRML or a DXF view of the geometry of a building product model, to create bills of materials in spreadsheet format, to generate batch input for an analysis program etc. The resulting view models can be further processed by specialised applications to produce final construction documents, or they can be used as additional information sources in a dedicated CAD environment.

Whilst their goals may be quite different, such tasks have certain common aspects w.r.t. product modelling and mapping:

1) For easy processing, the resulting data structures must correspond closely to the input expected by the targeted application(s);
2) The source data are as a rule heavily pruned, reducing the structure to a much simpler representation;
3) The process is *not intended to be reversible*.

This case study, adapted from the ToCEE project, illustrates such typical one-way mapping using as source model the fairly complex IFC Project Model to create a sample virtual reality presentation based on a very simple information structure. The examined problem addresses the geometric presentation of any tangible building objects defined in the IFC model through their *bounding boxes* by means of the *cube* construct of VRML (cf. Ames et al. 1996).*)

For conciseness, the IFC Project Model has been slightly modified here: the SELECT construct allowing both 2-dimensional and 3-dimensional reference coordinates is replaced with a direct link to *IfcAxis2Placement3D*, and all available representation options for building objects are reduced to *IfcBoundingBox*. The reference path from *IfcProduct* to *IfcBoundingBox* is left unchanged to preserve the modelling style as far as possible.

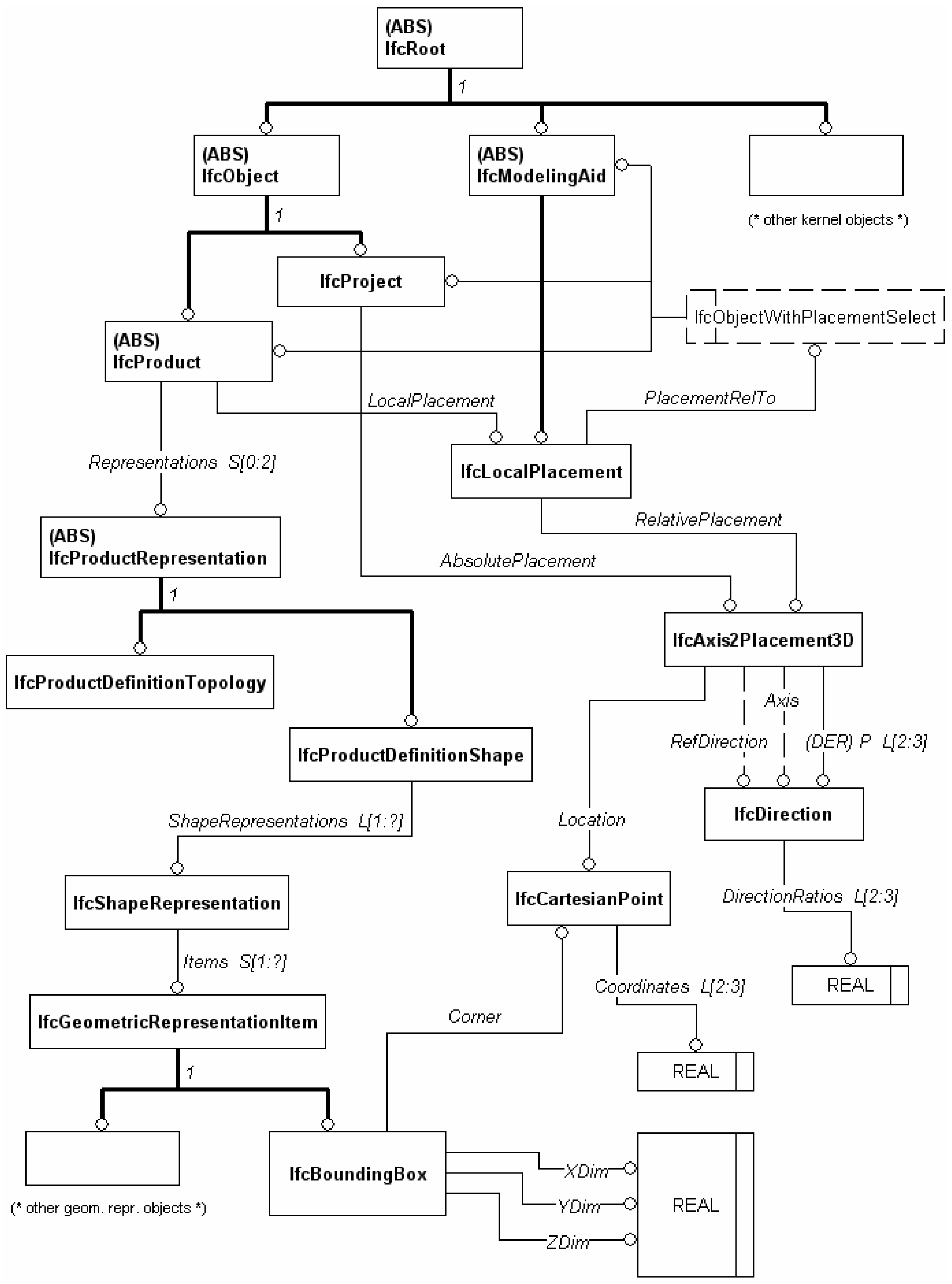
The sample target model contains only *one* entity, defining just a portion of the data that would be needed for a full virtual reality presentation of IFC-based product data according to the VRML specification. However, as simple as it is, this model is sufficient for a rough presentation of the geometry of all physical building objects defined in an IFC Project Model through their bounding boxes; since such representations are obligatory in IFC, it is in fact the fastest way to obtain a view of the geometry of a building. This can be quite useful for example in the early design phases.

The mapping exercise itself includes:

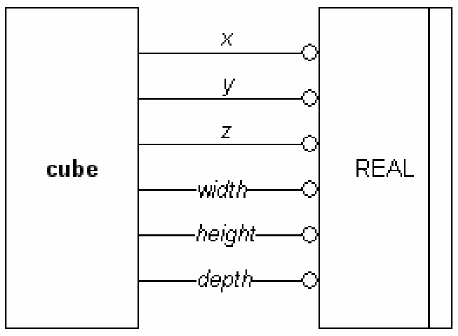
– a **pre-condition** to select the relevant entities for the mapping in the source model,
– the definition of a **complex geometric transformation**, and
– a **1:1 indirect mapping** of the attributes of **`IfcBoundingBox`** to a VRML **`cube`**.

*) Here it might be argued that because the required output format conforms to a known standard, and the source model is a standardised data model, it would be simpler and more efficient to use a straight-forward interface program, instead of a mapping specification processed by a generic mapping engine. However, even such standards – currently VRML 2.0 and IFC 2x - do not remain frozen, but continue to be developed, with little concern of each other. A mapping specification concentrating on the set relational description of the problem, and hiding any specific implementation details, should be easier to maintain, and after all the preferable approach.

Source schema **(part of `"IfcProjectModel"`):**



Target schema **(part of `"vrml_view_model"`) :**



Mapping specification:

```
(MAP vrml_view_model FROM SHARED IfcProjectModel
 CLASSES
  (MAP cube FROM IfcProduct
    CONDITIONS
      (PRED (IfcProduct HAS AT LEAST 1 Representations) )
    VAR
      (MAKE ?BBOX FROM
        (FOR ?S DO
          (AND (Items OF (Representations OF THIS) HAS VALUE ?S)
               (?S IS INSTANCE OF IfcBoundingBox) ) )
        )
      (MAKE ?TRANS
            CONSTRUCTOR coord-trans ARGS LocalPlacement)
      (MAKE ?P FROM (REF ?BBOX -> Corner FOR Coords) )
      (MAKE ?R FROM (MAPCAR (LAMBDA (X Y) (+ X Y))
                            ARGS ?P
                                 (LISTOF ?BBOX -> Xdim
                                         ?BBOX -> Ydim
                                         ?BBOX -> Zdim)) )
      (MAKE ?PG CONSTRUCTOR pt-trans ARGS ?P ?TRANS)
      (MAKE ?RG CONSTRUCTOR pt-trans ARGS ?R ?TRANS)
    ATTRIBUTES
      (MAKE x y z FROM ?PG)
      (MAKE width depth height
            FROM (MAPCAR (LAMBDA (U V) (- V U)) ARGS ?PG ?RG) )
    )
 PRESETS
  (LOAD "ifc-trans.lisp" FOR coord-trans pt-trans)
)
```

<u>Comments</u>:

The main part of this mapping task is the functional transformation needed to convert the points defining a bounding box in its local coordinate system within the IFC model to a reference point and three side lengths of a VRML cube in the global coordinate system of a VR world (see fig. 7.4).

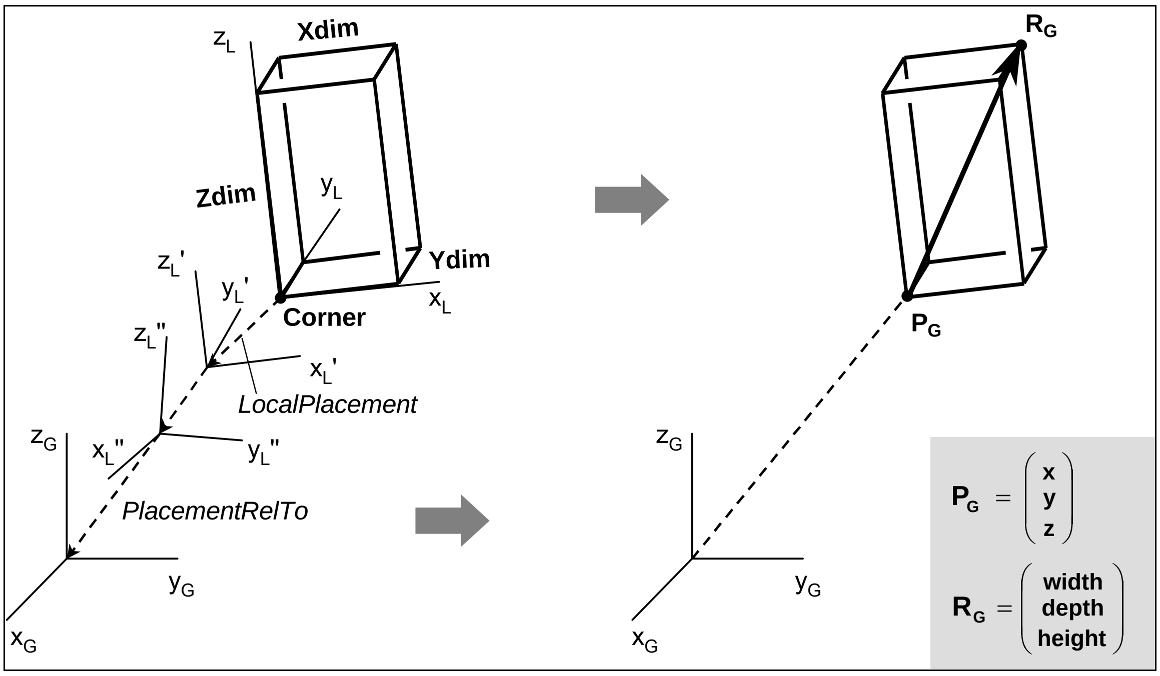

Xdim
Zdim
Ydim
z_L
y_L
x_L
z_L'
y_L'
x_L'
z_L''
x_L''
y_L''
Corner
LocalPlacement
PlacementRelTo
z_G
y_G
x_G
R_G
P_G
$P_G = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$
$R_G = \begin{pmatrix} \text{width} \\ \text{depth} \\ \text{height} \end{pmatrix}$

Fig. 7.4: Schematic presentation of coordinate definitions in the IfcProjectModel and in the sample target vrml_view_model

This transformation is quite complex, with pronounced algorithmic character. Its logic is to track first the link `IfcProduct.LocalPlacement = ↑L | L ∈ { IfcLocalPlacement }` in order to get the local coordinate system in which the geometry of an *IfcProduct* instance is described, and after that follow the relationship `L.PlacementRelTo = ( ↑M ∨ ↑P ∨ ↑T )` where `M ∈ { IfcModellingAid }, P ∈ { IfcProduct }, T ∈ { IfcProject }`, to determine the object whose coordinate system is used as reference. The second step is repeated until `L.PlacementRelTo = ↑T` is found, which is always defined in the global coordinate system. Fig. 7.5 shows schematically the necessary iteration procedure.

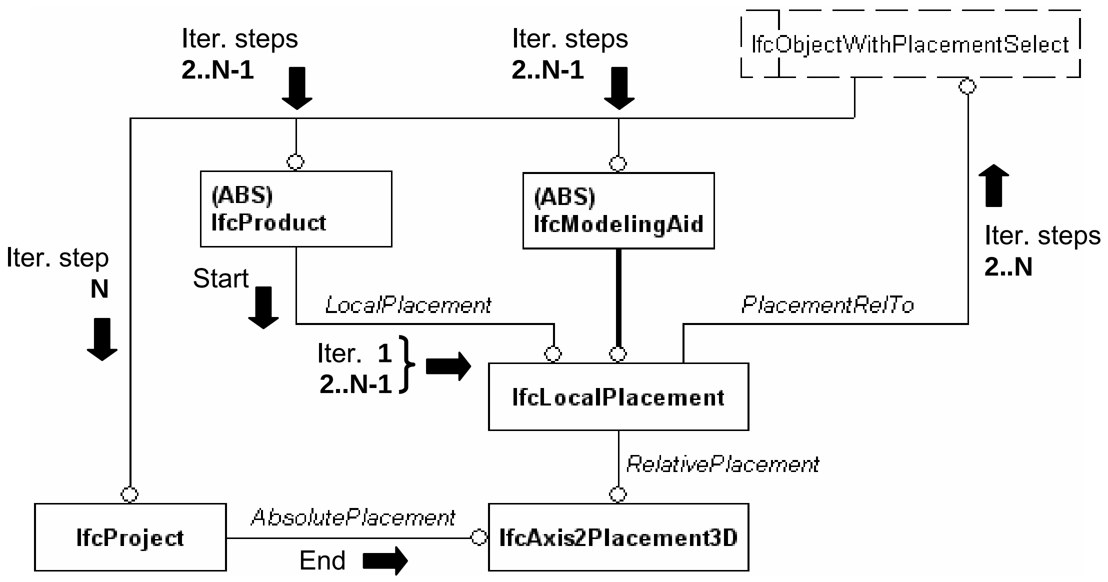
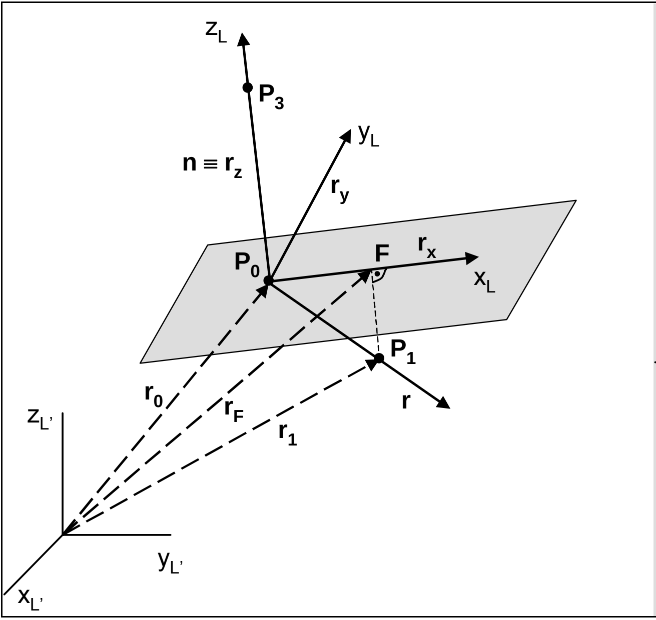


Fig. 7.5: Schematic presentation of the sequence of operations needed for the coordinate transformations from the IfcProjectModel to the sample target vrml_view_model

At each iteration step, the transformation matrix **R4**, representing translation and rotation in homogeneous coordinates, must be computed. This is done on the basis of the schema shown on fig. 7.6.



$\mathbf{P_0} \equiv$ *IfcAxis2Placement3D.Location*

$\mathbf{r} \equiv$ *IfcAxis2Placement3D.RefDirection*

$\mathbf{n} \equiv$ *IfcAxis2Placement3D.Axis*

$\mathbf{r_0} = \{ x_0\ y_0\ z_0 \}$

$\mathbf{r_1} = \{ x_1\ y_1\ z_1 \}$

$t_F = < \mathbf{n}.\mathbf{r_0} > - < \mathbf{n}.\mathbf{r_1} >$

$\mathbf{r_F} = \mathbf{r_1} + t_F.\mathbf{n}$

$\mathbf{r_x} = \mathbf{r_F} - \mathbf{r_0}$

$$\cos\varphi_1 = \frac{n_z}{|\mathbf{n}|}\ ;\ \cos\varphi_2 = \frac{n_y}{|\mathbf{n}|}\ ;\ \cos\varphi_3 = \frac{n_z}{|\mathbf{n}|}$$

$$\cos\theta = \frac{\mathbf{r_{x,x}} + t_F.\mathbf{n_x}}{|\ \mathbf{r_x}\ |}$$

$$\sin\theta = \sqrt{1-\cos^2\theta}$$

Fig. 7.6: Geometry schema used to determine the transformation matrix **R4** *for* L → L' *according to IFC 2.0 and ISO 10303-42*

From the formulae given in this figure, substituting

$A = \cos^2\varphi_1$,	$B = \cos^2\varphi_2$,	$C = \cos^2\varphi_3$,
$D = \cos\varphi_1.\cos\varphi_2$,	$E = \cos\varphi_2.\cos\varphi_3$,	$F = \cos\varphi_3.\cos\varphi_1$,
$P = \cos\varphi_1.\sin\theta$,	$Q = \cos\varphi_2.\sin\theta$,	$R = \cos\varphi_3.\sin\theta$ and $S = \cos\theta$,

and performing the necessary geometric transformations, **R4** can be written as:

$$\mathbf{R4} = \begin{bmatrix} A - AS + S & D - DS + R & F - FS - Q & 0 \\ D - DS - R & B - BS + S & E - ES + P & 0 \\ F - FS + Q & E - ES - P & C - CS + S & 0 \\ x_0 & y_0 & z_0 & 1 \end{bmatrix} \qquad (1)$$

Now the coordinates of each point $P_k^L = \langle x_k\ y_k\ z_k \rangle$ in the new reference coordinate system L' can be obtained from:

$$P_k^{L'} = P_k^L \cdot \mathbf{R4} \qquad (2)$$

By applying (2) consecutively on each iteration step, the coordinates in the global coordinate system (G) of each $P_k^L \in \{ P^L \}$ can be derived as follows:

$$\overline{\mathbf{R4}} = \prod_{i=1}^{n} \mathbf{R4}_i \qquad (3)$$

$$P_k^G = P_k^L \cdot \overline{\mathbf{R4}} \qquad (4)$$

where $\mathbf{R4_i}$ describes the transformation $L_{i-1} \rightarrow L_i$, with $L_0 = L$ and $L_n = G$.

In the required mapping, the above transformations are defined with the help of two external functions: **`coord-trans`**, performing the whole iteration process and computing

$\overline{\textbf{R4}}$ according to (3), and **`pt-trans`**, determining the points { P^G } according to (4). The source code for the first of these functions is not trivial, extending on more than 120 lines, whereas the second is less than 15 lines long and performs only a straight-forward *vector* x *matrix* multiplication to return a list of three reals [x y z] representing the coordinates of each point $P_k^G \in \{P^G\}$.

The mapping specification itself involves only one inter-class mapping declaration, **`IfcProduct` → `cube`** which can be re-used by inheritance in all the subclasses of `IfcProduct`, i.e. `IfcWall`, `IfcColumn`, `IfcBeam`, `IfcDoor` etc. This declaration consists of three parts:

– a pre-condition (lines 4-5),
– several variable initialisations, accomplished by using the described transformation functions (lines 6-22), and
– the actual attribute mappings (line 23-26).

The pre-condition utilises a knowledge-based template as predicate allowing to select only those instances in `IfcProjectModel` which contain at least one representation.

A knowledge-based function in the form of a search expression is used also as *initialiser* for the variable `?BBOX`, selecting from all possible representation items characterising an instance of a subclass of `IfcProduct` the one which is a bounding box (lines 7-11).

The other variables defined in the `VAR` section of the class mapping declaration are used to store intermediate results which are then applied to the attribute mapping transformations. `?TRANS` stores the transformation matrix computed with the help of the function `coord-trans`, which is passed one argument, `LocalPlacement`, needed for the first iteration step (lines 12-13). `?P` stores the `Corner` point of the bounding box (line 14), and `?R` stores the opposite point on its space diagonal (lines 15-19). Here the `LISTOF` term operator is used to collect in a list the attributes `Xdim`, `Ydim` and `Zdim` of the bounding box instance referenced through `?BBOX`. At last, `?PG` and `?RG`, computed with the help of the function `pt-trans`, are used to store the corresponding coordinates to `?P` and `?R` in the global coordinate system of the VR-model, (lines 20, 21).

After this long preparatory work, the actual arguments of `cube` can easily be obtained: `x`, `y` and `z` are copied from `?PG` using list destructuring (line 23), and `width`, `depth` and `height` are computed from the values of `?PG` and `?RG` using `MAPCAR` to subtract the individual coordinate values one by one (lines 24-25).

The whole process of populating the target model comprises 6 steps:

1) Reducing the set of source instances to the effect of the `CONDITIONS` statement;
2) Generating **`?BBOX`** and **`?TRANS`** for the remaining mapping domain;
3) Generating **`?P`** – depending on `?BBOX`;
4) Generating **`?R`** – depending on `?P`;
5) Generating **`?PG`** and **`?RG`** – depending on `?P`, `?R` and `?TRANS`;
6) Generating **`cube`** – depending on `?PG` and `?RG`.

In a more refined implementation, some of these steps can probably be combined to reduce the number of scans of the mapping domain. Due to the relatively short implementation time and the academic character of the development effort, this has not been done in the prototyped version of the mapping engine by the author.

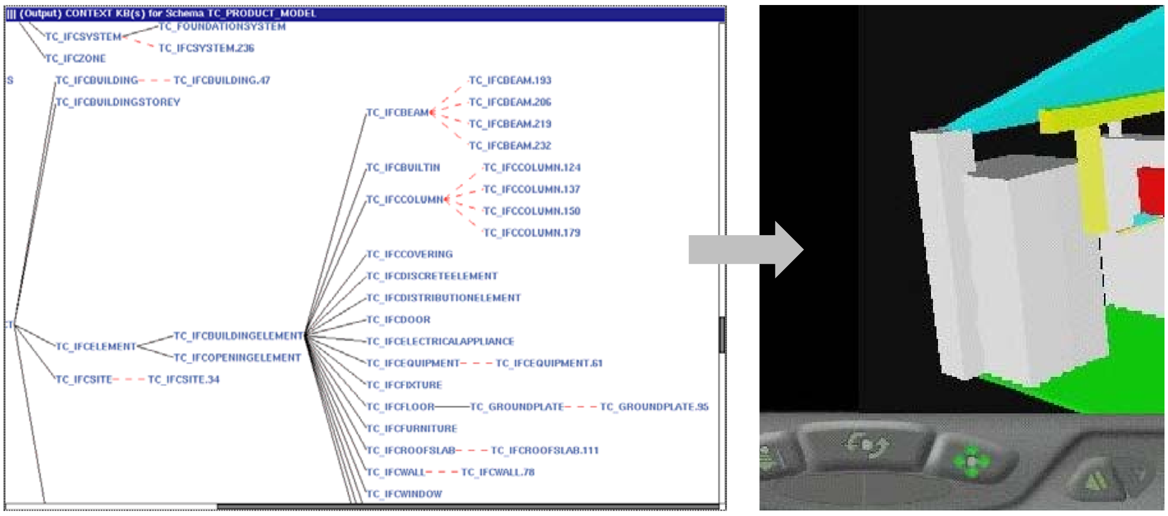


Fig. 7.7: Presentation of IFC-based model data taken from the demonstration example of the ToCEE project on the GUI desktop of the project data server (left) and after mapping to VRML (right)

7.4 Discussion

The three case studies examined in this chapter complete the presentation of the concepts for tackling the semantic interoperability problems in the proposed concurrent engineering environment. Along with being to some extent of practical value to the mentioned projects COMBI and ToCEE, the performed studies have served the following objectives:

1) to examine the representational potential of CSML for typical engineering tasks within the scope of the proposed environment;
2) to gather experience of the process of mapping modelling itself;
3) to measure the performance of the mapping process with respect to the separate constructs of CSML, and finally,
4) to compare CSML with other developed mapping approaches on a more practical basis.

The presented studies cover only a part of the practical testing of CSML performed in the course of work. For example, cooperative design tasks have been examined also for the design interaction chains “detailed structural analysis – foundation analysis”, “preliminary architectural design – preliminary structural design – preliminary HVAC design”; views of the COMBI building model and of IFC-based product data have been generated also on DXF basis as wire-frame and boundary representation models etc.

There were no problems in any of the performed tasks that CSML could not deal with. However, in the course of work some of the test cases evoked modifications and improvements to certain CSML constructs, especially when some sophisticated mapping patterns became more clear. For example, there have been a few changes to the syntax and semantics of *terms* and *term operators* to make the constructs in which they are used more consistent, and some of the *template operators*, most notably the `assoc_op`, have repeatedly been subject to changes in the course of the verification process.

In its current form, used in the examples of this and the preceding chapters, CSML covers well the mapping problems identified in chapter 5. Some constructs can be improved in future w.r.t. their readability as it was not always readily clear what the respective actions of the mapping engine will be. Also, in all examined cases additional hand-coded external functions were needed to tackle some of the required transformations. Whilst the use of such external functions for the algorithmic part of a mapping is an intentional feature of CSML, it would nevertheless be worth considering to enhance the scope of the CSML operators, so that some more standard arithmetic computations can be covered.

The technical part of the modelling and specification process itself proved to be less difficult than expected. For each of the presented case studies, it took only a few hours to develop the necessary specifications. Of course, this does not include the coding and testing of the involved external functions, neither the process of understanding the semantics of the source and target schemas, as they were in each case already familiar to the author. However, exactly this preliminary work should not be underestimated because in real-world cases it is expected to take a large amount of time and involve considerable efforts of experts from different domains. It is actually the best justification of the need for a high-level mapping language, allowing to use semantic constructs that are close to the problem domain and are at the same time well separated from the details of the implementation.

The run-time performance of a mapping was found to depend significantly on the quality of the mapping specifications. Experiments with alternative ways of expressing some of the required mapping transformations showed that there can be considerable divergence in the performance (up to 50%), especially when set relational operations are involved. Most careful consideration were found to require the complex template operators, such as `ASSOC` and `NEW`, as well as the used knowledge-based templates. Appropriately designed external functions did, on the other hand, always speed up the mapping process. However, the usage of such functions must be weighed against the inevitably degraded readability and maintainability of the mapping model.

As a whole, the run-time characteristics of the mapping process showed the expected analogy on technical level to the generation of database views in multidatabase systems. Therefore, a closer study of advanced implementation methods from database research can be of great benefit for future improvements of the overall mapping process.

Finally, along with a review on more theoretical basis, the conducted studies helped to make a qualitative assessment of CSML w.r.t. other developed mapping approaches. For this purpose, the practical solution of the case study presented in section 7.1 was analysed, along with a task from the COMBI project performed with the help of XP-RULE (mapping of the COMBI Neutral Building Model to the STEP AP 201”Explicit Draughting” -- ISO 10303-201 1994) and examples from (EPM 1996) for EXPRESS-X[*)]. A valuable source for the undertaken comparison was found also in (Amor 1997).

The characteristic features of the examined approaches are summarised in table 7.1 below.

[*)] The case study given in section 7.1 has been used as test bed for most of the examined approaches; the results of their survey are discussed in detail in (Liebich et al. 1995) and (Verhoef et al. 1995).

Table 7.1 Qualitative estimation of examined mapping approaches

Features	Transformr	EXPRESS-M	EXPRESS-V	EXPRESS-X	XP-RULE	OM	EDM-2	ACL/KIF	VML	**CSML**
General characteristics										
Level of support of specific mapping problems for CEE	L	L	L	M	M	M	M/H	H	M	H
Modularity of the approach	L	L	L	L	M	M	M	L	M	H
Plug-in support	–	–	–	–	L	–	L	–	–	M
Interactive mapping support	–	–	–	–	–	–	M	M/H	M/H	M
Language characteristics										
Language level	M	M	M	M	M/H	M	M	M	M/H	M/H
Declarative/Procedural style: D/P	P	D/P	P	D/P	D	D	D	D	D	D
Object-oriented language support	–	L	L	L	L	L	M	M	H	H
Object-oriented method support	–	–	–	–	–	–	–	L	L	–
Representational characteristics										
Conditional mapping support	L	M	M	M/H	H	M/H	H	M	H	H
Handling of aggregations/associations	L	M	M	M/H	M/H	M/H	X	M	M/H	H
Handling of object/attribute relationships	M	M	M	M/H	M/H	M/H	X	M	H	H
Explicit/Implicit type handling: E/I	E	E	E	E	E	E	E	I	E/I	E/I
Unit handling	–	L	–	L	–	–	X	M	–	X
Object creation	L	M	M	M	M	M	M	L	M	n/a
Attribute initialisation	L	M	M	M	M	M/H	M	L	M/H	H
Temporary structures (Yes/No)	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes
Bi-directional mapping support	–	–	L	L	L	L	L	L	M	n/a
Development environment										
Development tools	–	–	–	–	M	M	M	–	M	L
Graphical modelling support	–	–	–	–	–	M	L	–	M	L
Graphical browser (Yes/No)	No	No	No	No	No	Yes	Yes	No	Yes	Yes

(**Key**: L = Low, M = Medium, M/H = Medium to high, H = High, X = only with external functions)

As shown in this table, none of the surveyed approaches (including CSML) provides the full range of requirements suggested in chapter 5. For example, all EXPRESS-based languages appear to be tightly glued with the EXPRESS modelling paradigm, and some are even conceived as extensions to EXPRESS which can lower considerably the required modularity in a distributed environment. EDM-2 and VML also seem to be quite closely linked to the specific environment architectures in which they are used, and ACL/KIF relies on highly intelligent agents which makes it suitable mainly in pure knowledge-based environments.

In summary, it can be said that there does not yet exist an ultimate solution to all the semantic interoperability problems that can appear in distributed PDT-based environments. However, in spite of their limitations, any of the examined approaches can be useful in certain situations, depending on the specific needs. For example, as pointed out in (Verhoef et al. 1995), if the goal is the interoperability in a highly interactive, tightly integrated environment, VML, ACL/KIF, EXPRESS-V, EXPRESS-X and XP-RULE may be suitable choices. If compatibility with STEP/EXPRESS is of primary importance, the languages developed as part of STEP (EXPRESS-M, EXPRESS-C, EXPRESS-X) would be preferable. If the development process of product data models, including schema evolution and schema migration, is the main focus of work, languages like EXPRESS-M and EXPRESS-X would be best suited. If a knowledge intensive design environment with sophisticated application tools is being envisioned, EDM-2, VML or XP-RULE should be given preference, and platforms that do not support mappings explicitly, but provide flexible dynamic object evolution features (Hakim 1993; Cleetus 1995; Kowalczyk 1997) would be worth considering.

However, all examined approaches, except for CSML, assume that an integrated shared model must be maintained by the IT system. Most efforts address mappings to/from that assumed model, trying to keep it in a continuously consistent state. Whilst CSML can also support such approach, it has been shown that it is not necessarily the desired architecture for an enabling environment for concurrent engineering. Therefore, the efforts in the development of CSML have been directed especially towards supporting the modularity of the proposed overall approach, allowing clear separation of the suggested services, and platform-independent implementation of the individual components. At the same time, a CSML specification can (but need not) incorporate other advanced features, like knowledge-based expressions and foreign function code, and can itself be seamlessly integrated in client-server communication to support the solution of run-time interoperability issues, such as consistency checking, code checking, conflict management etc.

In this way, CSML fulfils the major requirements for a mapping language in CEE.

Chapter 8: Prototype Project Data Server for Concurrent Engineering

. . . *and an act hath three branches;*
it is, to act, to do and to perform.
– William Shakespeare, Hamlet

To validate the developed concepts presented in chapters 3 to 7, a prototype project data management server (PROMISE) and selected small example clients have been implemented as part of this thesis. A modified version of the server, with slightly simplified functionality, but extended with some practical features, has been successfully tested within the frames of the ToCEE project. Core features of the server have been utilised also in the COMBI project, although in COMBI only file-based data exchange and passive WWW communication have been addressed. Details of the ToCEE server implementation from an end-user point of view are given in (Katranuschkov & Hyvärinen 1998) and (Hyvärinen et al. 1999). The overall architecture and the prototyped knowledge-based features are discussed in (Scherer & Katranuschkov 1999).

This chapter presents an overview of the design principles that have governed the realisation of the server, the rationale for the chosen *frame-based representation paradigm* for the implementation platform and the main features of the developed application and user interfaces.

At the end, practical issues related to building design are discussed on the basis of a larger case study adapted from the ToCEE project.

8.1 Basic Concepts

In chapter 3 the basic requirements to the CEE system were identified, and a principal client/server environment architecture was outlined.

In chapter 4 the basic client/server software model and the major systemic interoperability problems to be tackled by the project data server of the CEE system were discussed, and in chapter 5 an approach for the management of semantic interoperability problems related to the non harmonised model world of AEC was proposed.

By looking at these problems from the viewpoint of software development, the following design considerations w.r.t. the implementation of the project data server can be identified:

1) The server should not be limited to pure data management tasks, but should be capable to support advanced functionality, taking over various coordination and cooperation activities.
2) The components of the server architecture (Request Broker, Data Management Server, stored procedures and agents) have distinctly different goals and scope, operate on different levels of the modelling framework, and “percept” different portions of the objective reality of the IT system.

3) The support of the system ontology requires explicit representation not only of modelling object instances, but also of modelling object classes and meta model concepts.
4) The support of advanced functionality involves symbolic and rule-based processing features that go beyond the capabilities of typical object systems.

In accordance with these considerations, the following major design decisions have been taken:

Modular approach
The components of the server architecture are separately implemented, and are linked by means of a generalised communication protocol, presenting a light-weight version of the communication model discussed in chapter 4, stripped off access control and some other meta information not needed in the static component model of the server. In this way, a possibility for a distributed implementation on the basis of different programming paradigms has been provided.

Hybrid representation
The Object Request Broker operates on the level of the system ontology and the communication model of the framework. Its basic functionality is related to the management of concurrent requests, the identification of access rights etc. This functionality does not require knowledge-based processing capabilities, but has its focus on concurrency. Therefore, an implementation on the basis of Java has been selected (Harold 1997).
In contrast, the data management server and the related server agents operate on the level of detailed project data models and have to support both basic data access functionality and more advanced interoperability methods. Therefore, an implementation on the basis of the frame-based paradigm (Minsky 1975) on top of the Knowledge Engineering Environment KEE (Intellicorp 1994) has been selected. A comprehensive description of the frame-based paradigm is provided e.g. in (Fikes & Kehler 1985) and (Cunis 1992).

Whilst the implementation of the Request Broker does not involve any specific software features of greater interest, the architecture of the actual project data server PROMISE (Project Model Based Information Server) deserves some more attention. The specific aspects of its realisation are outlined in the following section.

8.2 Server Architecture

The chosen representation paradigm enables the realisation of a server architecture that resembles closely the architecture of typical expert systems (see fig. 8.1). However, in contrast to expert systems, designed to assist engineers in the solution of specific problem solving tasks, the purpose of the knowledge-based methods of the project data server are to react to general-purpose, yet complex queries and assertions, and to adapt dynamically to a variety of changes in the project models resulting from the concurrent, simultaneous work of the separate professionals in a design team.

The main component of the proposed server architecture is its *knowledge base* containing a set of defined data model schemas and the respective instantiated models. Each data model schema in the knowledge base is organised in a frame structure, and contains exclusively *class frames*, roughly corresponding to the entity structuring given in the EXPRESS-C schema of the model. In contrast, the instantiated models contain only *instance frames* with inherited attributes and behaviour from the respective underlying data model schema.

All frames can act both as “normal” objects, responding to messages from the outside and/or from other objects, as well as to participate in rule-based queries and assertions.
The *interface module* has the task to resolve properly the requested operations by remote clients, and to activate the appropriate server object methods responsible for the execution of the client requests. All incoming requests and the respective responses are aligned with the formal EXPRESS-C specifications in the data models, and are represented in accordance with the adopted Information Container formalism. In this way, the interface module hides from the clients the implementation details of the server, which simplifies client implementation.

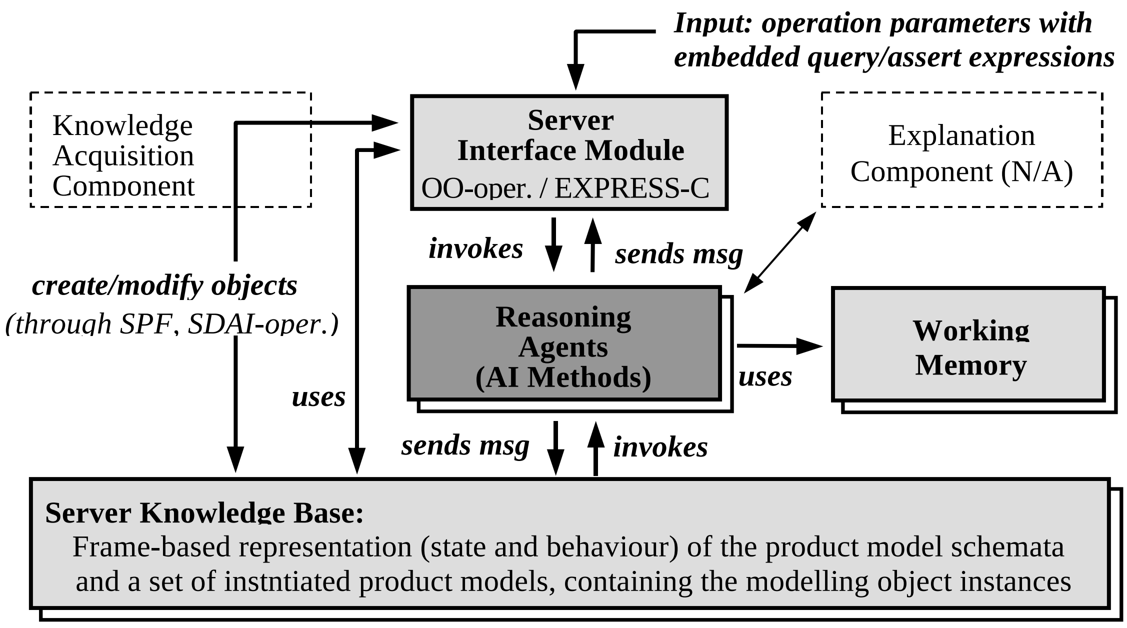

Input: operation parameters with embedded query/assert expressions
Knowledge Acquisition Component
Server Interface Module
OO-oper. / EXPRESS-C
Explanation Component (N/A)
invokes
sends msg
create/modify objects
(through SPF, SDAI-oper.)
Reasoning Agents (AI Methods)
uses
Working Memory
uses
sends msg
invokes
Server Knowledge Base:
Frame-based representation (state and behaviour) of the product model schemata and a set of instntiated product models, containing the modelling object instances

Fig. 8.1: Principal architecture of the project data server

The *reasoning agents* perceive changes in the environment and react to such changes with appropriate actions. Within the server architecture each reasoning agent is invoked by an object-oriented method whenever a knowledge-based function is addressed. This can be done *explicitly*, by executing a respective operation, such as[*]:

```
IfcRelUsesProducts.execMeth
  ("find"
    searchExpr:"(FOR ?X DO (?X HAS AT LEAST 2 RelatedObjects))")
```

or happens *automatically*, when the state of an attribute monitored by a demon method is changed.

Typically a reasoning agent will retrieve the required data from the knowledge-base together with the appropriate reasoning rules and will place them in the working memory of the server. After that the inference process on the retrieved data and rules is started, and, on its completion, the result is returned to the calling method. Currently, the server implementation applies the inference engine of the KEE system including forward and backward chaining as well as different search algorithms. These methods can be further enhanced using the same component architecture.

[*] The exact syntax of the function is slightly adapted for readability.

The remaining (dashed-box) components shown in fig. 8.1 are given only for comparison with typical expert system architectures. An explanation component is difficult to realise for a server software, and a comprehensive knowledge acquisition component has been beyond the scope of the server implementation, although new knowledge can (and is) continuously introduced in the knowledge base through respective object-oriented “assertions”.

8.3 Application Interfaces

The methodology to define and execute operations was discussed in chapter 4. Concise definitions of all prototyped operations are provided in appendix V. The implemented prototype environment and the principal schema of the remote execution of operations are illustrated on fig. 8.2.

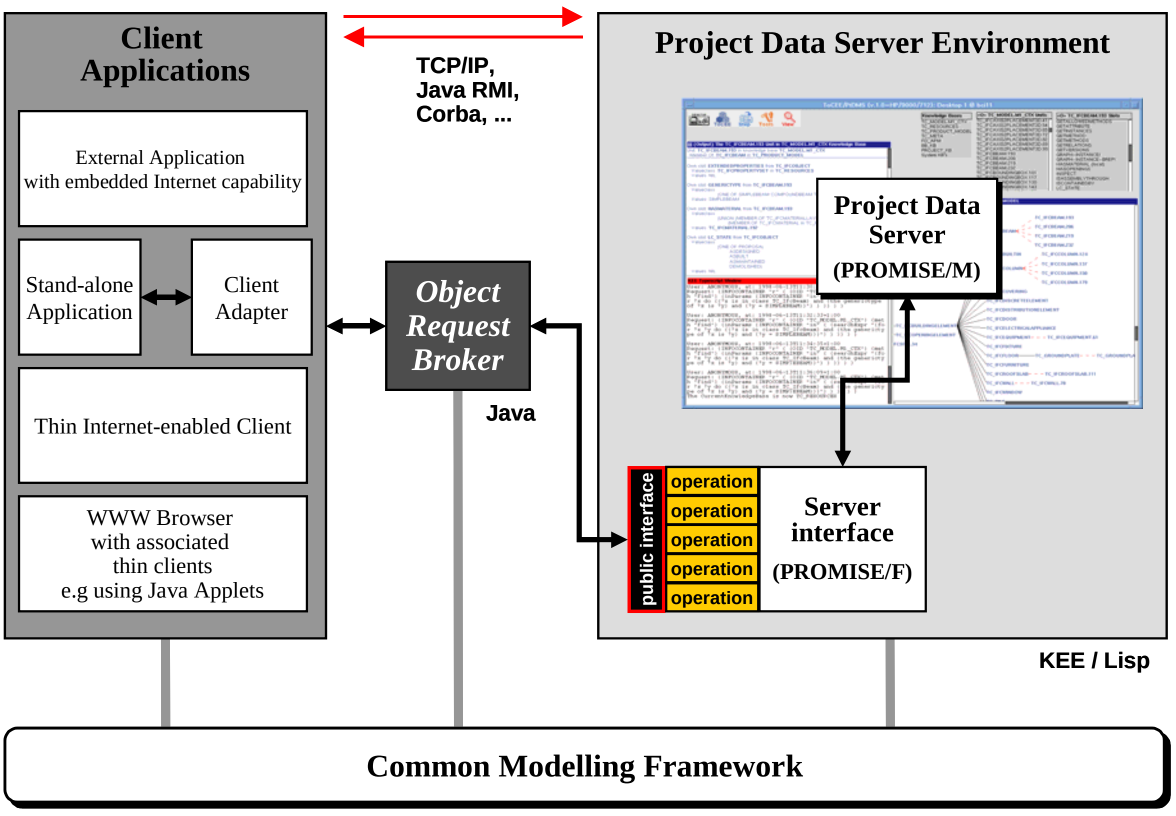

Client Applications
External Application with embedded Internet capability
Stand-alone Application
Client Adapter
Thin Internet-enabled Client
WWW Browser with associated thin clients e.g using Java Applets
TCP/IP, Java RMI, Corba, ...
Object Request Broker
Java
Project Data Server Environment
Project Data Server (PROMISE/M)
public interface
operation
operation
operation
operation
operation
Server interface (PROMISE/F)
KEE / Lisp
Common Modelling Framework

Fig. 8.2: Overview of the implemented client/server CEE system

On the basis of the prototyped operations, sample client adapters have been implemented for a structural analysis program (fig. 8.3) and for a foundation design system (fig. 8.4). With the provided formalism to define operations, the development of these two adapters has been a matter of days.

Currently, the API of the prototype implementation of PROMISE encompasses more than 50 operations that can be used by remote client applications (see appendix V). Most of them are adapted from the SDAI specification (ISO 10303-22 1998) or support data exchange functionality on the basis of STEP physical files (ISO 10303-21 1994). More interesting are some of the interoperability model-level operations.

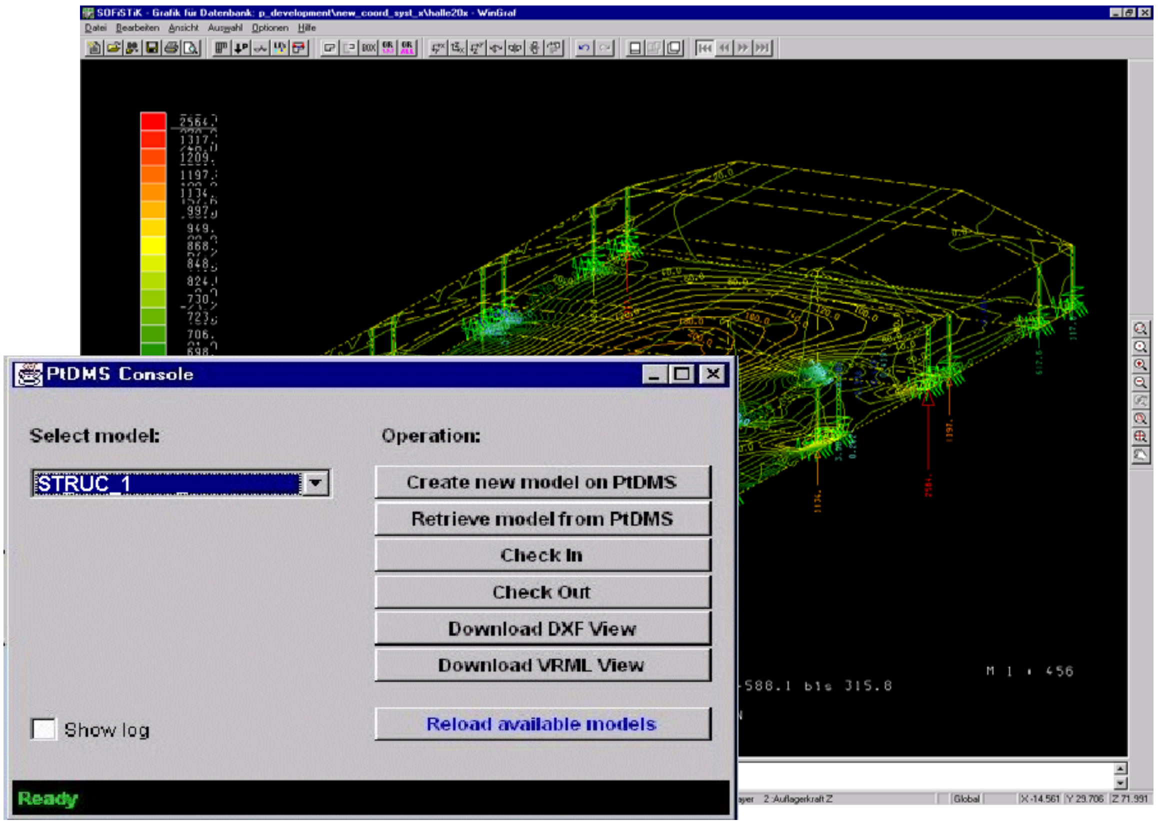

SOFiSTiK - Grafik für Datenbank: p_development\new_coord_syst_x\halle20x - WinGraf
Datei Bearbeiten Ansicht Auswahl Optionen Hilfe
PtDMS Console
Select model:
STRUC_1
Operation:
Create new model on PtDMS
Retrieve model from PtDMS
Check In
Check Out
Download DXF View
Download VRML View
Show log
Reload available models
Ready
M 1 : 456
-588.1 bis 315.8

Fig. 8.3: *Sample client adapter for structural analysis*

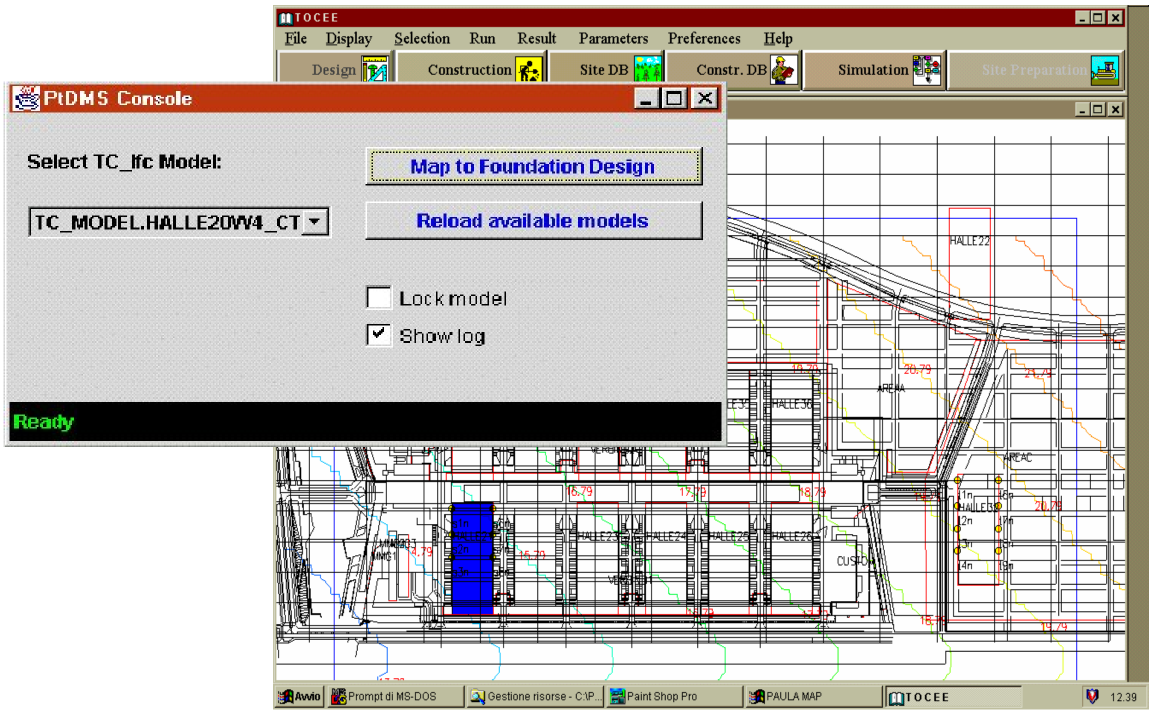


Fig. 8.4: *Sample client adapter for foundation design*

8.4 Interoperability Services

The basic interoperability services in the current prototype implementation of the project data server include *check in*, *check out*, *map*, *match* and *merge*.

Check in / check out are the principal operators that introduce the concept of long transactions. Otherwise, their implementation is trivial. The actual data transfer is performed on the basis of STEP physical files.

"Check in" enforces a *CheckIn* state, and "check out" enforces a *LongWrite*.

Mapping is a complex process involving two data models, a mapping schema and a source model as input to produce a new target model as output. It is comprised of the following principal steps:

1) Parsing of the specifications;
2) Creating the mapping domain on the basis of the class level specifications and conditions imposed on the source objects;
3) Creating the mapping extent on the basis of the mapping domain, the cardinalities of the involved mapping templates and conditions imposed on the target, or involving at least one entity from the target;
4) Solving of direct equivalences, for which the result set has been unambiguously determined;
5) Reassessment of unsolved equivalences because of deferred operations, such as NEW;
6) Pruning of duplicate instances from the target model.

A mapping always creates a new model version and enforces a *CheckIn* state.

Matching involves two models based on the same schema. Principally, it is a simple procedure, but it strongly depends on the method by which object identification is maintained.

Thus, if unique object IDs are maintained by the server on model level, it is easy to find all corresponding objects and compare the attributes.

In contrast, if globally unique IDs are maintained, the objects in two model versions will formally not be the same (different IDs). In this case a full scan of the data is necessary, making the process convoluted and slow.

The output of a matching operation are the changed objects found. New model versions are not created and the model states are not changed. The matching process itself requires exclusive data access and is therefore performed in a temporary workspace to which the models are first copied.

Merging is a process which is in principle very similar to a *join operation* in a RDBMS. However, even if all types of *joins* are considered, it is unlikely that an automatic solution can be obtained. Therefore an interactive version is suggested, by which all possible conflicts are reconciled by the end-users as shown on table 8.1[*)].

[*)] Currently there is no convenient implementation of such interactive operations with PROMISE. The procedure is not aligned with the general Information Container and communication model specifications, but is realised with dedicated code at the client side. Also, although the merging process may involve more than two models, it is performed pair-wise. This is due to the fact that the simultaneous coordination of *any* combination of multiple models in an interactive procedure is a very difficult implementation issue.

Table 8.1: Merging policies and respective actions in interactive merging mode

Merging Policy	**Action**				
	Same object in both models	Modified object, no conflicts	Modified object, conflicting data	Object only in base model	New object in ref. model
Interactive restrictive	Yes	Yes	Notify & Negotiate	Notify & Negotiate	Notify & Negotiate
Interactive left join	Yes	Yes	Notify & Negotiate	Yes	Notify & Negotiate
Interactive right join	Yes	Yes	Notify & Negotiate	Notify & Negotiate	Yes
Interactive union	Yes	Yes	Notify & Negotiate	Yes	Yes

Fig. 8.5, redrawn from fig. 3.7, provides an idea of the inter-relationships of the interoperability operators in the suggested project coordination process. However, the involved combinations can be quite complicated and require much further research.

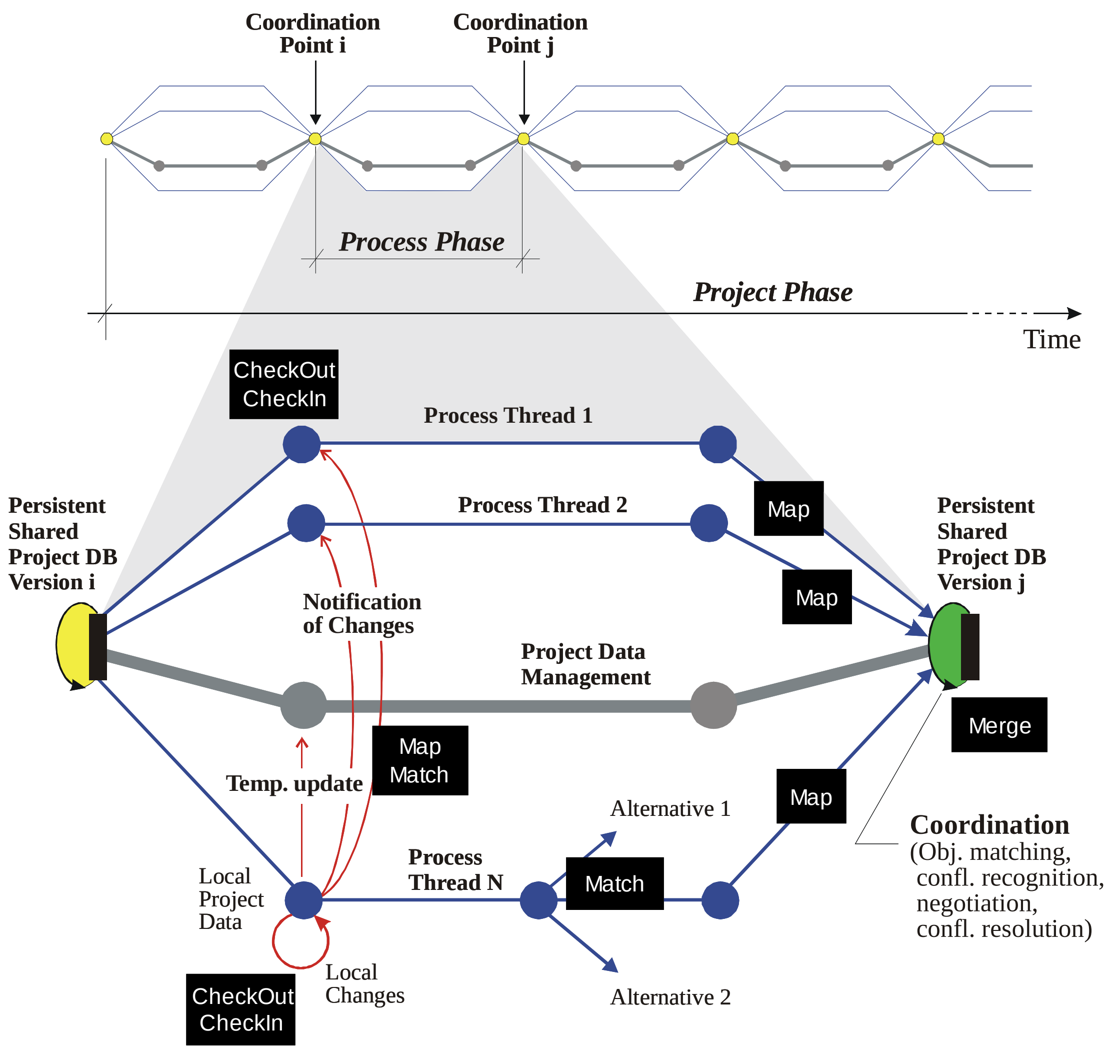
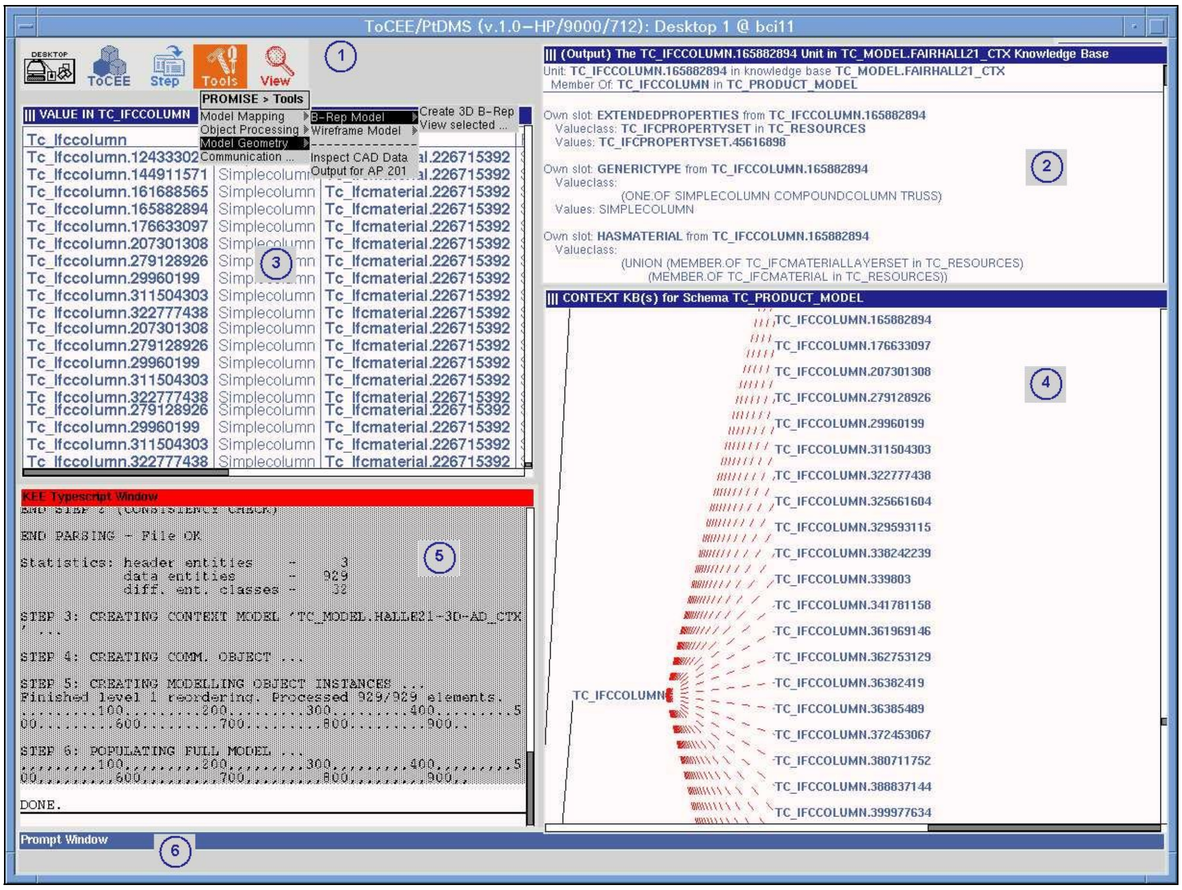


Fig. 8.5: Principal schema of the use of interoperability services for project coordination

8.5 User Interface

Normally, local server operations are limited to simple configuration and maintenance tasks which do not require a specific graphical user interface. Typical examples for such cases include the Java RMI registry (Harold 1997) and some http-servers. However, complex server programs, such as database servers, require more advanced administration which is often supported by a GUI. For similar purposes, PROMISE enables password protected local use for privileged project data management or system administration.

The "local" mode differs from the normal "server" mode of operation in the following way: (1) the access rights are set to "superuser" giving full permission to use all available system functions as well as full read/write access to all models, (2) maintenance operations are enabled which allows to start/stop the server, stop running processes, load/unload model schemas and model data, lock models for remote use, modify access rights etc., and (3) the model data can be manipulated both with the help of the public server operations (as in "server" mode) and interactively, with the help of a GUI as shown on fig. 8.6 below.



||| VALUE IN TC_IFCCOLUMN

Tc_Ifccolumn		
Tc_Ifccolumn.12433302		al.226715392
Tc_Ifccolumn.144911571	Simplecolumn	al.226715392
Tc_Ifccolumn.161688565	Simplecolumn	Tc_Ifcmaterial.226715392
Tc_Ifccolumn.165882894	Simplecolumn	Tc_Ifcmaterial.226715392
Tc_Ifccolumn.176633097	Simplecolumn	Tc_Ifcmaterial.226715392
Tc_Ifccolumn.207301308	Simplecolumn	Tc_Ifcmaterial.226715392
Tc_Ifccolumn.279128926	Simp	Tc_Ifcmaterial.226715392
Tc_Ifccolumn.29960199	Simp	Tc_Ifcmaterial.226715392
Tc_Ifccolumn.311504303	Simplecolumn	Tc_Ifcmaterial.226715392
Tc_Ifccolumn.322777438	Simplecolumn	Tc_Ifcmaterial.226715392
Tc_Ifccolumn.207301308	Simplecolumn	Tc_Ifcmaterial.226715392
Tc_Ifccolumn.279128926	Simplecolumn	Tc_Ifcmaterial.226715392
Tc_Ifccolumn.29960199	Simplecolumn	Tc_Ifcmaterial.226715392
Tc_Ifccolumn.311504303	Simplecolumn	Tc_Ifcmaterial.226715392
Tc_Ifccolumn.322777438	Simplecolumn	Tc_Ifcmaterial.226715392
Tc_Ifccolumn.279128926	Simplecolumn	Tc_Ifcmaterial.226715392
Tc_Ifccolumn.29960199	Simplecolumn	Tc_Ifcmaterial.226715392
Tc_Ifccolumn.311504303	Simplecolumn	Tc_Ifcmaterial.226715392
Tc_Ifccolumn.322777438	Simplecolumn	Tc_Ifcmaterial.226715392

||| (Output) The TC_IFCCOLUMN.165882894 Unit in TC_MODEL.FAIRHALL21_CTX Knowledge Base

```
Unit: TC_IFCCOLUMN.165882894 in knowledge base TC_MODEL.FAIRHALL21_CTX
  Member Of: TC_IFCCOLUMN in TC_PRODUCT_MODEL

Own slot: EXTENDEDPROPERTIES from TC_IFCCOLUMN.165882894
   Valueclass: TC_IFCPROPERTYSET in TC_RESOURCES
   Values: TC_IFCPROPERTYSET.45616898

Own slot: GENERICTYPE from TC_IFCCOLUMN.165882894
   Valueclass:
            (ONE.OF SIMPLECOLUMN COMPOUNDCOLUMN TRUSS)
   Values: SIMPLECOLUMN

Own slot: HASMATERIAL from TC_IFCCOLUMN.165882894
   Valueclass:
            (UNION (MEMBER.OF TC_IFCMATERIALLAYERSET in TC_RESOURCES)
                (MEMBER.OF TC_IFCMATERIAL in TC_RESOURCES))
```

||| CONTEXT KB(s) for Schema TC_PRODUCT_MODEL

KEE Typescript Window

```
END STEP 2 (CONSISTENCY CHECK)

END PARSING - File OK

Statistics: header entities    -     3
            data entities      -   929
            diff. ent. classes -    32

STEP 3: CREATING CONTEXT MODEL 'TC_MODEL.HALLE21-3D-AD_CTX
' ...

STEP 4: CREATING COMM. OBJECT ...

STEP 5: CREATING MODELLING OBJECT INSTANCES ...
Finished level 1 reordering. Processed 929/929 elements.
.........100.........200.........300.........400.........5
00.........600.........700.........800.........900..

STEP 6: POPULATING FULL MODEL ...
.........100.........200.........300.........400.........5
00.........600.........700.........800.........900..

DONE.
```

Fig. 8.6: Screenshot of the server desktop

(1) main menu bar; (2) output window showing the browsing of an object instance in text format; (3) output window showing the browsing of the model data in table format; (4) output window showing the browsing of the inheritance structure of a model; (5) log window for server activities; (6) prompt window.

The main menu of the GUI contains options to start/stop the server, to examine and to stop running processes, as well as to use the functions of the server API. Fig. 8.7 shows a screenshot of the pull-down menu enabling the selection of a server operation from the GUI. The menu is created dynamically, depending on the referenced data model.

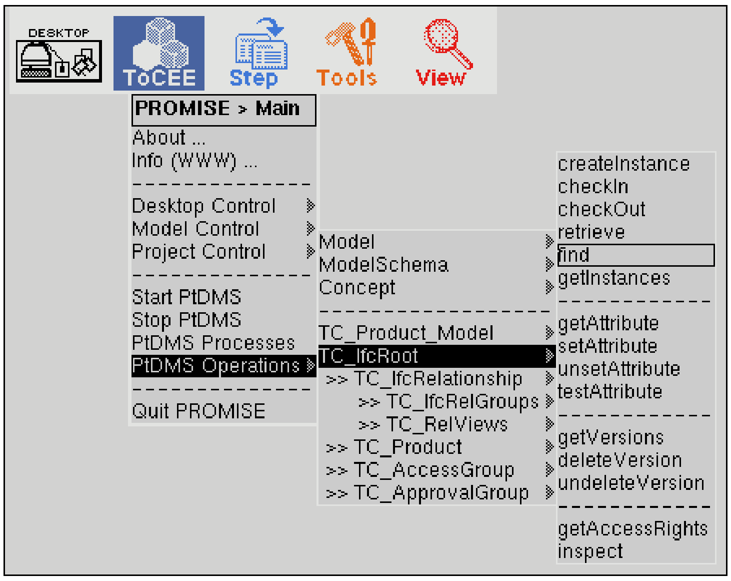


Fig. 8.7: Screenshot of the main menu showing the dynamically assigned data management operations imported from the environment ontology (adapted from ToCEE)

The main menu itself is adaptable as shown on fig. 8.8 below.

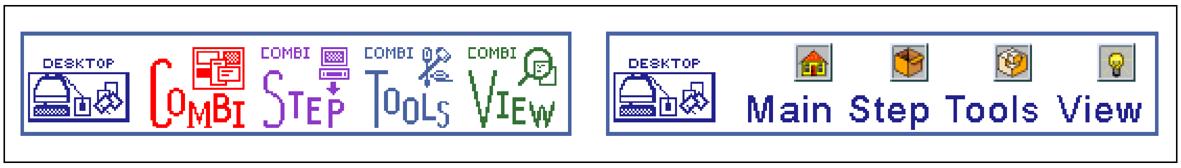


Fig. 8.8: Alternative versions of the main menu of the GUI of PROMISE
/left: adapted for the COMBI system, *right:* basic (default) version /

The GUI provides also some additional features accessible through the other submenus of the main menu.

- The “Step” submenu allows to import/export EXPRESS schemas and local STEP physical files.
- The “Tools” submenu provides facilities for examining a model mapping (fig. 8.9), for executing a mapping, for creating/editing/deleting objects etc.
- The “View” submenu enables browsing of the data in different ways, e.g. by dynamically navigating down the object hierarchy (fig. 8.10), as a full inheritance graph, in tabular form etc.

In addition to these basic functions, each model, class, instance or attribute has an associated context menu which is accessible through the mouse.

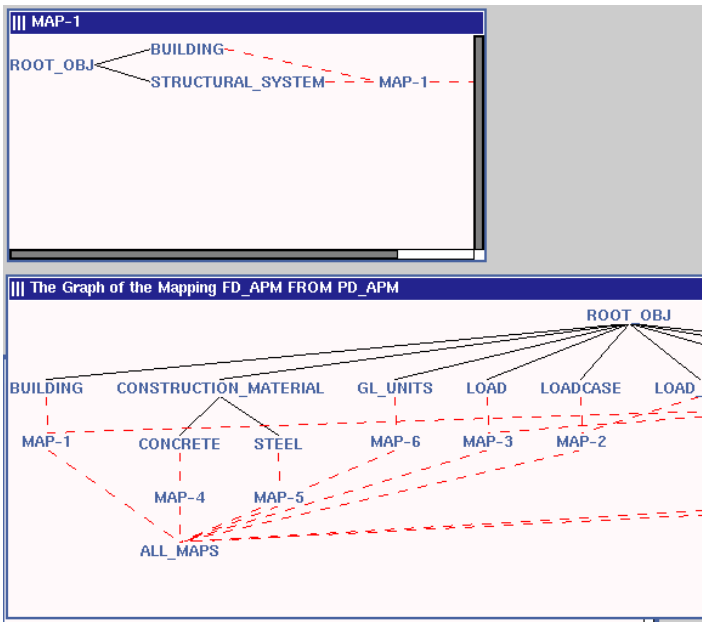


Fig. 8.9: Graphical examination of a mapping schema

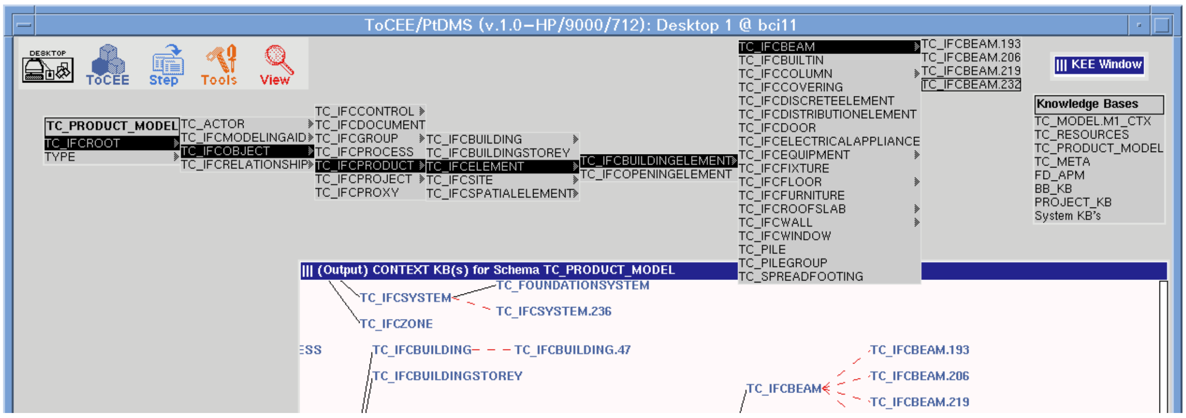


Fig. 8.10: Cascading navigation through a project model with the GUI of PROMISE

8.6 Implementation Example

This section demonstrates in a larger case study the features of the prototyped project data server (PROMISE) in its interplay with other software tools developed in the ToCEE project.

Unlike the previous sections, in which many screenshots of the server platform and the developed sample clients were presented, in this section the server behaves as a typical "server" i.e. in the background. The provided example, adapted from the final ToCEE workshop, depicts the solution of a *typical cooperative design task*. The case building presents a simplified version of Hall 21 of the New Munich Fair facility. The focus of the design activities is on the solution of a conflicting situation, due to an (unexpected) late design change.

As a starting point, it is assumed that the building owner requests a change at a certain stage in the design of the building. The specific task is to provide light crane equipment with working area of the crane extending over the whole length of the hall (see fig. 8.11).

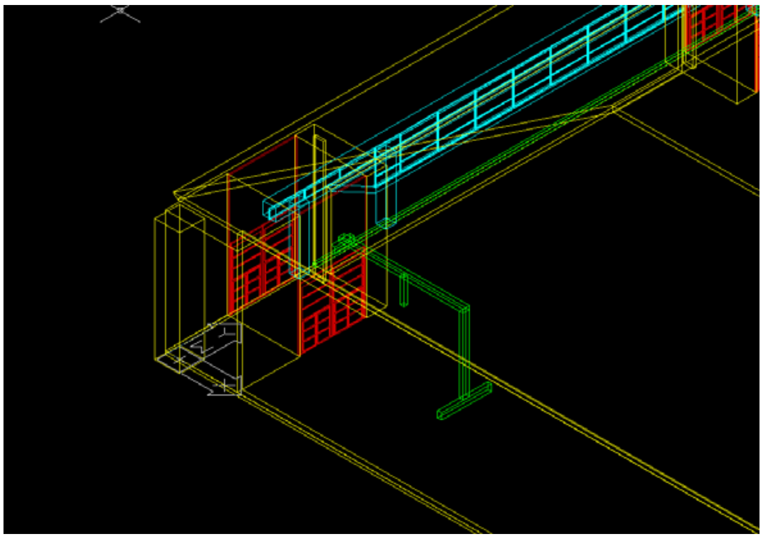


Fig. 8.11: Initial design change for the example demonstration scenario

Although the order of the owner goes to the architect, this task cannot be fulfilled by the architect alone. Specialist knowledge of the whole design team (HVAC, electrical, structural, foundation engineer etc.) is needed in order to consider properly the possible consequences of the design change. Therefore, in parallel to determining the crane type and location, the architect sets up new concurrent work tasks as shown schematically on fig. 8.12.

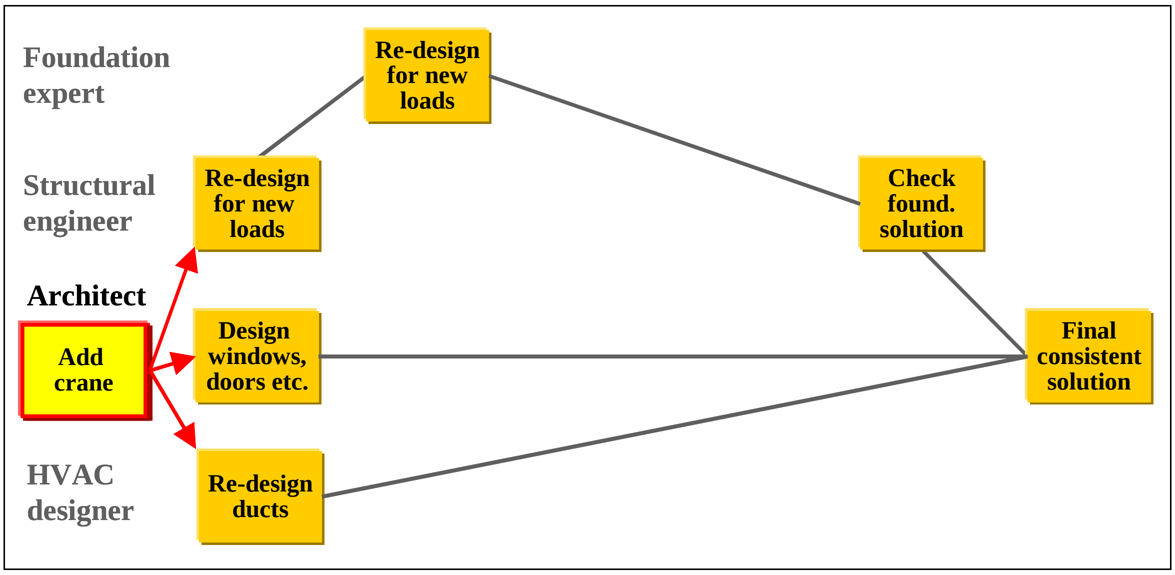


Fig. 8.12: Initial workflow of the example demonstration scenario

To focus the discussion let us now concentrate on a specific **conflict between the architectural and the HVAC design** which arises in the course of the parallel project work.

According to the set up workflow, the task of the HVAC designer is to re-design the duct system for ventilation and air conditioning so that it fits to the proposed change by the architect. However, this single work task (from the point of view of the architect) involves a complex sequence of actions (from the point of view of the HVAC designer) requiring the use of a variety of tools, system services and types of data.

This **expanded view of the work task of the HVAC designer** is shown on fig. 8.13 below.

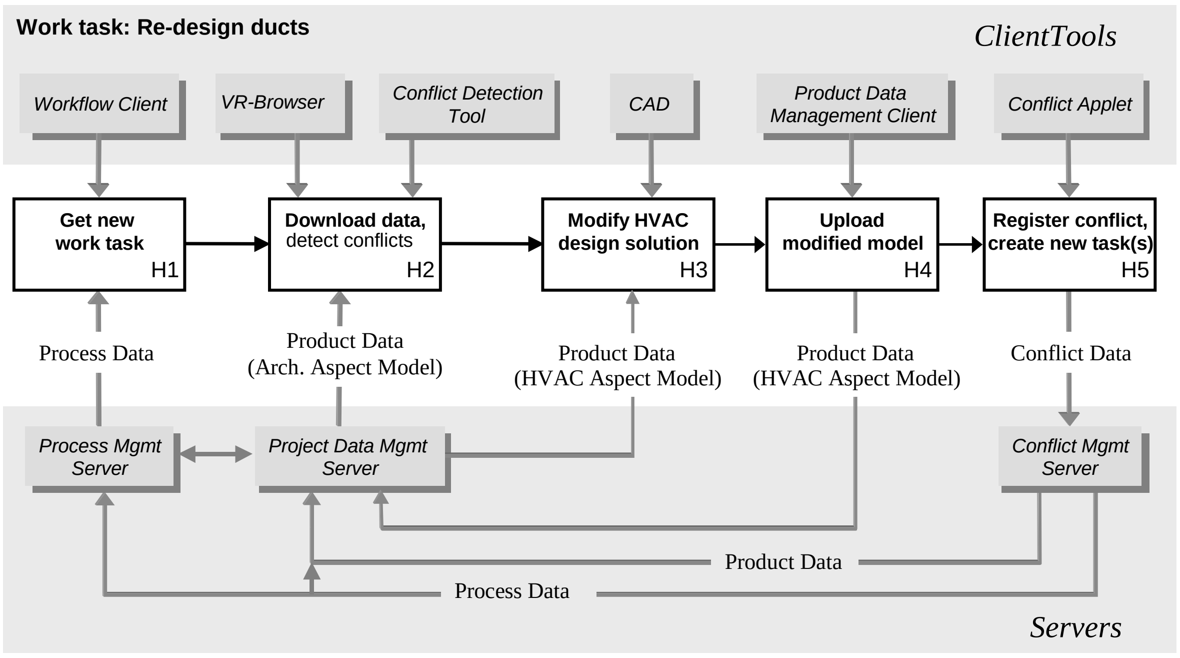

Work task: Re-design ducts
ClientTools
Workflow Client
VR-Browser
Conflict Detection Tool
CAD
Product Data Management Client
Conflict Applet
Get new work task H1
Download data, detect conflicts H2
Modify HVAC design solution H3
Upload modified model H4
Register conflict, create new task(s) H5
Process Data
Product Data (Arch. Aspect Model)
Product Data (HVAC Aspect Model)
Product Data (HVAC Aspect Model)
Conflict Data
Process Mgmt Server
Project Data Mgmt Server
Conflict Mgmt Server
Product Data
Process Data
Servers

Fig. 8.13: Sequence of actions in the example work task "Re-design ducts" of the HVAC designer

At first the HVAC designer gets informed of the new task through a *workflow client* (step H1).

After that he checks out the current, up-to-date project data from the architectural domain (or aspect) model, and then uses e.g. a virtual reality tool for fast inspection of the changes made by the architect (step H2). This can be done locally, or by using the server "view" operation.

From this inspection it becomes obvious that the crane presents a **potential problem**, because, when moving, it may collide against the ventilation ducts in the hall. A detailed analysis with ToCEE's *specialised conflict detection tool* confirms the suspected problem (step H3).

The conflict detection tool has been designed to detect geometric conflicts for objects moving along pre-defined working paths, and is implemented on top of Autodesk's Architectural Desktop. Fig. 8.14 gives an example of its use, showing selected frames from the animation produced after the analysis of the crane path.

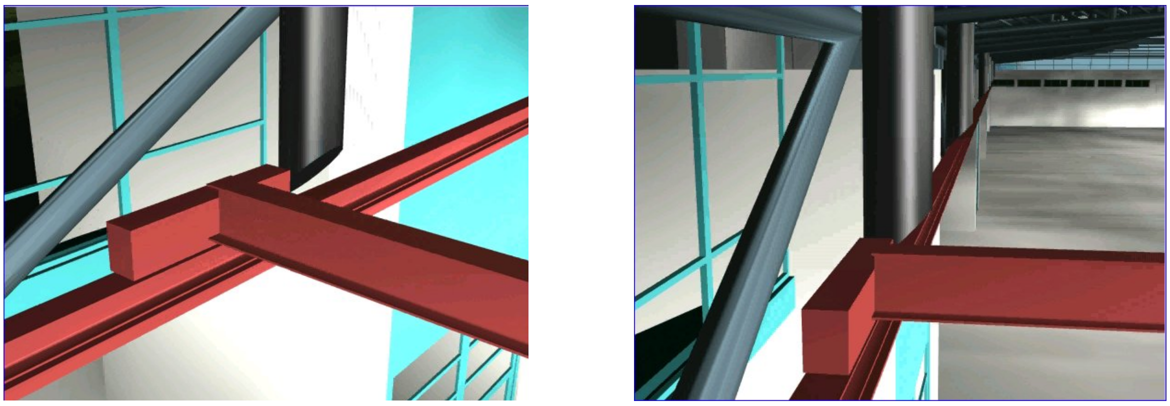


Fig. 8.14: Detection of geometric conflicts with moving objects

Due to this, now obvious conflict, the HVAC designer is not able to fulfil the requirements of the architect as suggested. Instead, he proposes an **alternative solution**, modifying respectively the project data with the help of his *CAD* system (step H4).

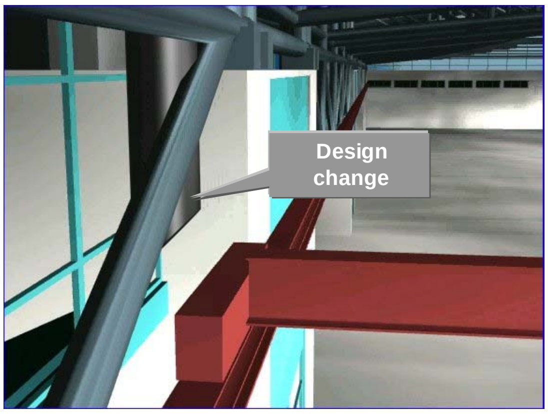


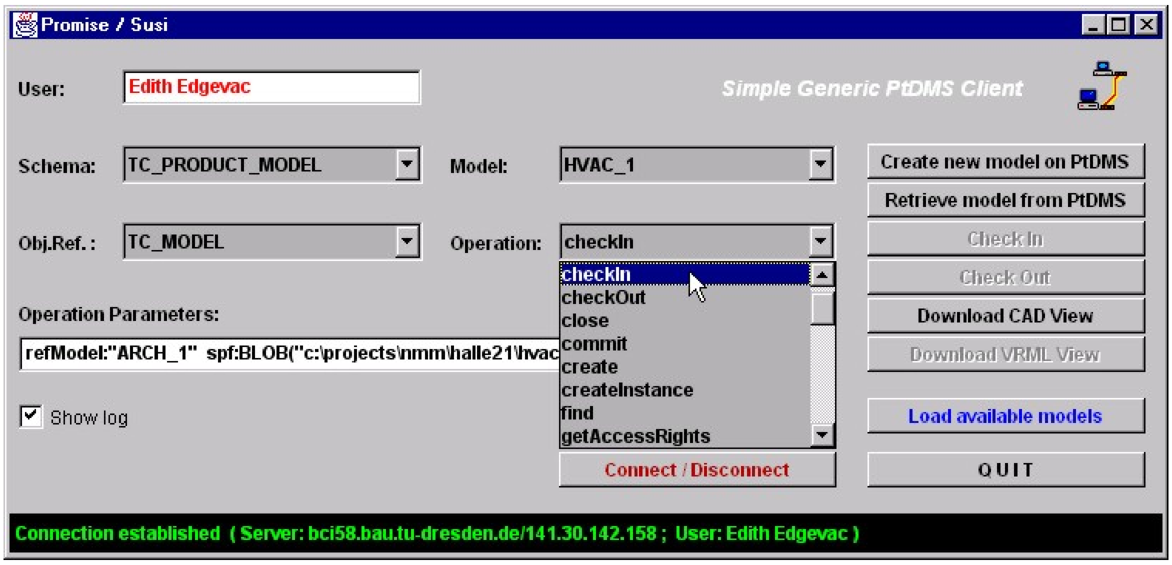
Fig. 8.15: Change of the design data due to the geometric conflict between crane and ventilation ducts

In a conventional approach this would now complete the task of the HVAC designer. He would then notify the architect about the conflict with an informative message and will either attach his alternative proposal as a drawing file to his e-mail message or, by more advanced organisational and IT infrastructure, he will store this file on the document management system used in the project. It might also be possible to exchange the product data with the architect, but the co-ordination of the process of conflict resolution will still happen only in the heads of the designers, without notable IT support.

However, with the help of the prototyped services, a more **rigorous approach to change and conflict management** is possible.

After the project data is modified to represent the proposed new design alternative, the HVAC designer uploads the new version of the HVAC aspect model to *PROMISE.*

This can be done either by a specialised client tool, similar to the client adapter shown on fig. 8.3, or with the generic light-weight project data management client *PROMISE / Susi* as shown on fig. 8.16.


Promise / Susi

Simple Generic PtDMS Client

User: Edith Edgevac

Schema: TC_PRODUCT_MODEL

Model: HVAC_1

Obj.Ref.: TC_MODEL

Operation: checkIn

Operation Parameters:

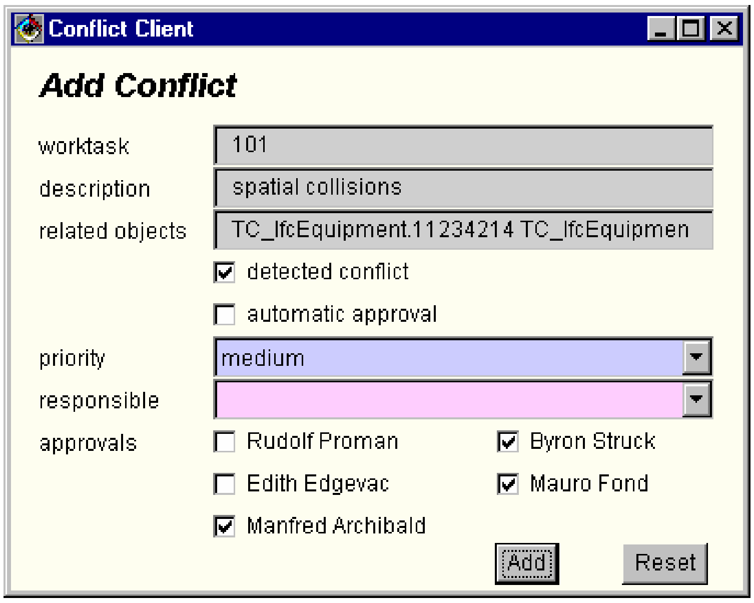
refModel:"ARCH_1" spf:BLOB("c:\projects\nmm\halle21\hvac

Show log ✓

Connection established (Server: bci58.bau.tu-dresden.de/141.30.142.158 ; User: Edith Edgevac)

Fig. 8.16: Uploading the HVAC aspect model data to the project data server with the help of the generic project data client PROMISE / Susi

Then the HVAC designer registers the conflict on the *Conflict Management Server* with the help of a dedicated *Conflict Management Client* (see fig. 8.17), or, optionally, with a *Conflict Applet*, invoked from his WWW-Browser (step H5).


Conflict Client

Add Conflict

worktask	101
description	spatial collisions
related objects	TC_IfcEquipment.11234214 TC_IfcEquipmen
	✓ detected conflict
	☐ automatic approval
priority	medium
responsible	
approvals	☐ Rudolf Proman ✓ Byron Struck
	☐ Edith Edgevac ✓ Mauro Fond
	✓ Manfred Archibald

Fig. 8.17: Use of a conflict management client to report a conflict

The ***Conflict Management Server*** stores the conflict data including all related references to processes and project data objects in a central conflict database which enables the monitoring of the conflict status, the maintenance of the data consistency and integrity and the notification of all actors involved in the conflict solution. Along with that, the process management system automatically creates a new work task for the architect to negotiate the conflict.

The architect has now the responsibility to **react to the raised conflict**. The actual activities comprising this, initially not planned work are depicted on the expanded view of the work task given on fig. 8.18 below.

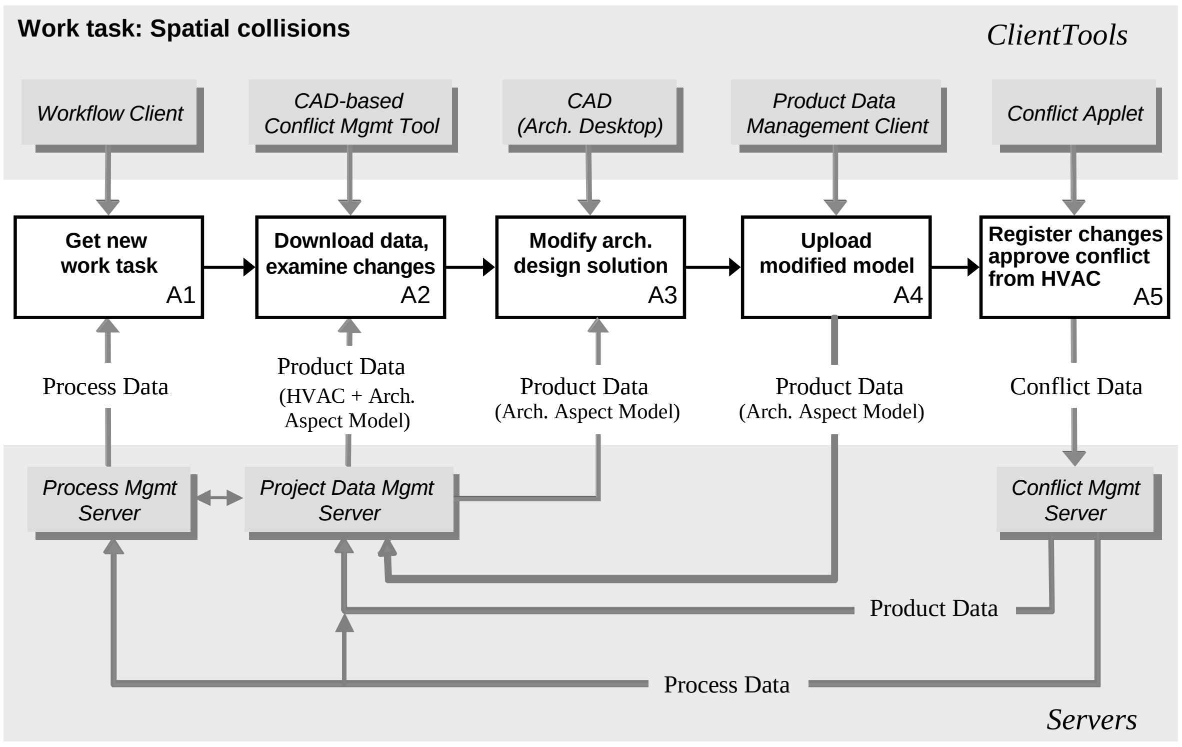

Work task: Spatial collisions
ClientTools
Workflow Client
CAD-based Conflict Mgmt Tool
CAD (Arch. Desktop)
Product Data Management Client
Conflict Applet
Get new work task A1
Download data, examine changes A2
Modify arch. design solution A3
Upload modified model A4
Register changes approve conflict from HVAC A5
Process Data
Product Data (HVAC + Arch. Aspect Model)
Product Data (Arch. Aspect Model)
Product Data (Arch. Aspect Model)
Conflict Data
Process Mgmt Server
Project Data Mgmt Server
Conflict Mgmt Server
Product Data
Process Data
Servers

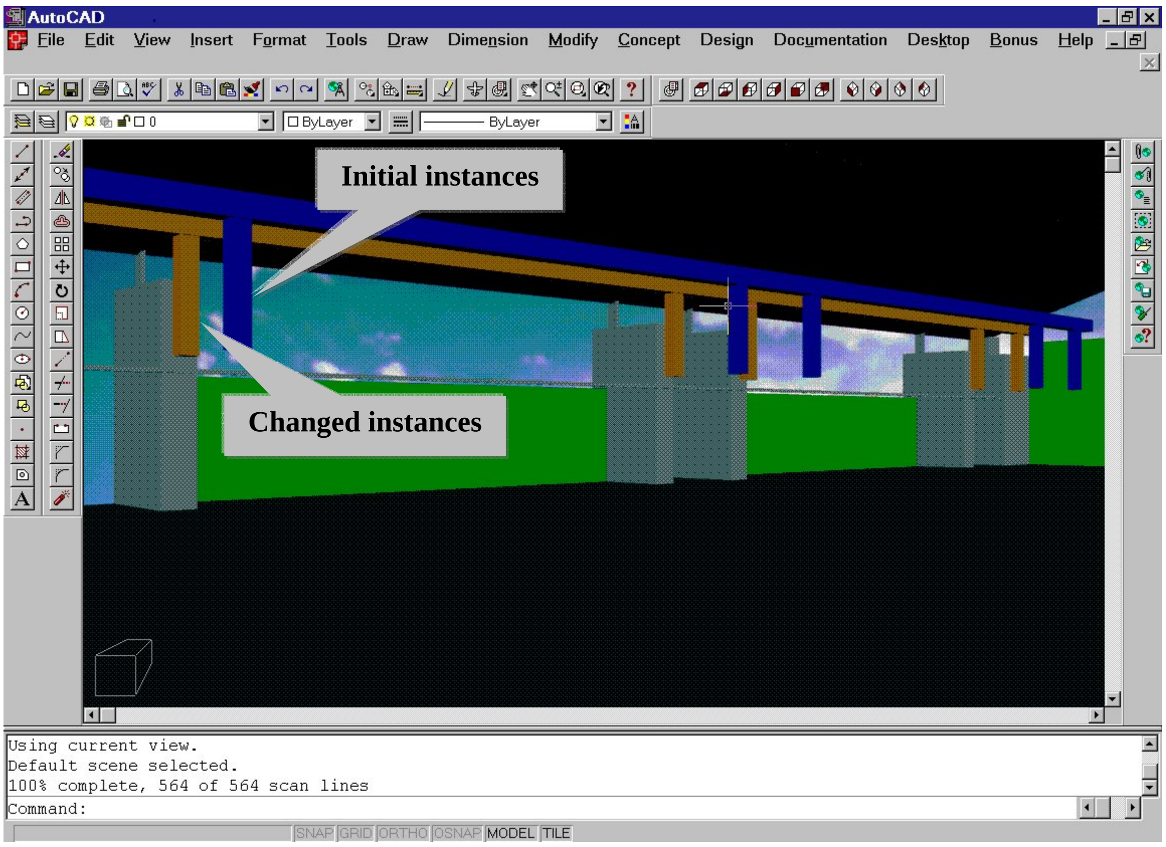
Fig. 8.18: Sequence of actions in the new work task of the architect, due to the detected conflict

The next action is to **examine the proposed changes** of the HVAC designer (step A2).

This can be done in a similar way as already described. Alternatively, the architect can use directly with his CAD system the ToCEE *change management tool* developed on top of Autodesk's Architectural Desktop which enables to view and compare the data of two model versions (fig. 8.19).

The analysis of the differences between the two models relies on the centrally maintained by the project data server object identifications for all project objects in all domain models. The tool supports direct queries to the project data server through the *InfoContainer API* described in this thesis. In order to visualise the changes, only the architectural model has to be loaded completely, whereas the new and changed objects in the HVAC model are obtained directly through a `match` operation.

After examining the proposed changes the architect has to **coordinate the solution of the conflict**. Assuming that he himself can agree with the modifications proposed by the HVAC designer, he must nevertheless take care that all other designers also agree with the modifications and re-work their design solutions accordingly. For this purpose he stops the running workflow with the *Process Wizard* and consequently sets up the **workflow for the needed re-design work**.


AutoCAD
File Edit View Insert Format Tools Draw Dimension Modify Concept Design Documentation Desktop Bonus Help
ByLayer
ByLayer
Initial instances
Changed instances
SNAP GRID ORTHO OSNAP MODEL TILE

```
Using current view.
Default scene selected.
100% complete, 564 of 564 scan lines
Command:
```

Fig. 8.19: The ToCEE change management tool embedded in Autodesk's Architectural Desktop

At this point the workflow has the form presented on fig. 8.20 below.

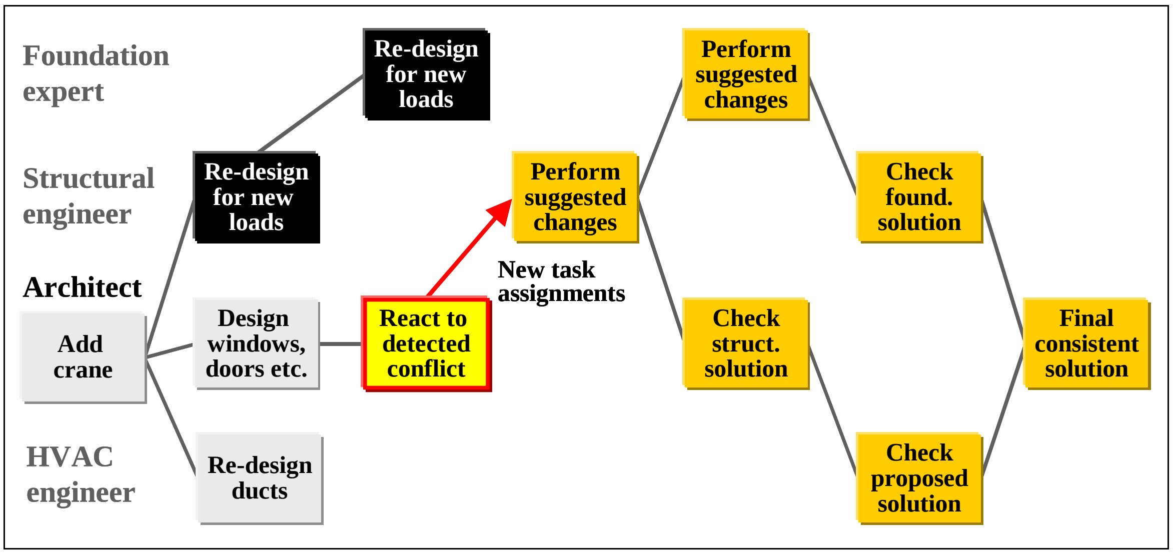
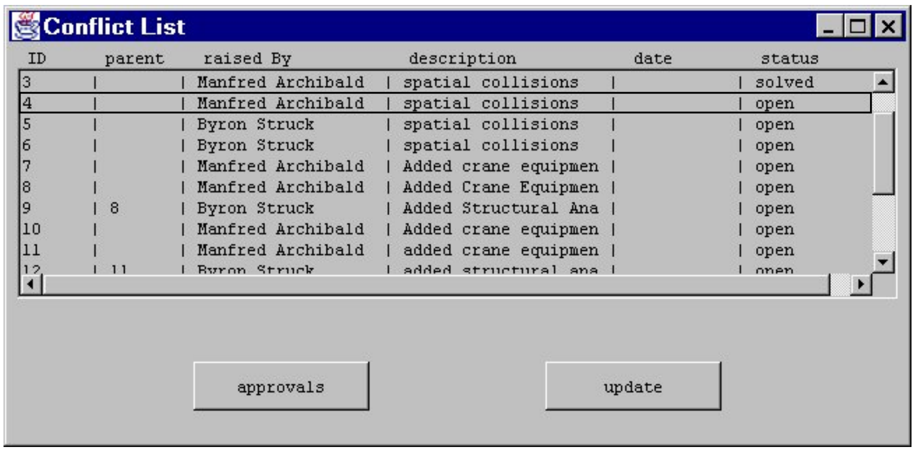


Fig. 8.20: Revised workflow including activities for conflict resolution (black boxes show aborted tasks, light grey boxes show tasks which are not aborted, as they are already finished)

After this coordination work the architect can adjust his own solution with the help of his CAD system (step A3) and then upload the changed model back to the project data server (step A4) in a similar way as already described for the HVAC designer. Finally, he approves the changes proposed by the HVAC designer as shown on fig. 8.21 (step A5).



```
ID      parent    raised By           description              date          status
3       |         | Manfred Archibald | spatial collisions   |             | solved
4       |         | Manfred Archibald | spatial collisions   |             | open
5       |         | Byron Struck      | spatial collisions   |             | open
6       |         | Byron Struck      | spatial collisions   |             | open
7       |         | Manfred Archibald | Added crane equipmen |             | open
8       |         | Manfred Archibald | Added Crane Equipmen |             | open
9       | 8       | Byron Struck      | Added Structural Ana |             | open
10      |         | Manfred Archibald | Added crane equipmen |             | open
11      |         | Manfred Archibald | added crane equipmen |             | open
12      | 11      | Byron Struck      | added structural ana |             | open
```

Fig. 8.21: Conflict approval

The updated architectural model can now be used as basis for modifying all other discipline-specific models by the other designers. Their work will typically follow a similar to the presented procedure, using the appropriate services of the environment.

A unique feature supported by the prototyped project data server is that it does not inhibit the existence of **temporarily inconsistent data** in order to restrict as little as possible individual and simultaneous work. Many of the developed services and tools contribute to the solution of such conflicting situations. Their usage, briefly outlined in the presented scenario, is summarised in the following table.

Table 8.2: Software tools used in the implementation example

Conflict management issues	Tools	Developed by
• Parallel work (time) conflicts, process coordination	• Process Management Server • Process Wizard • Workflow clients	external external external
• Distributed data consistency • Distributed data access • Change visualisation	• **Project Data Server (PROMISE)** • Generic project data client • **PROMISE** + CAD-Client	the author the author the author + external
• Conflict management, data coordination • Conflict notification and approval • Conflict detection	• Conflict Management Server + **PROMISE** • WWW-enabled conflict management clients • Conflict detection tool • Change management tool embedded in Autodesk's Arch. Desktop	external with participation of the author external external

However, whilst the abovementioned tools provide a functional user interface and many useful stand-alone features, their full power is manifested only through their coherent use in the CEE system. For example, in order to enable the functionality of the *conflict clients* and the individual work on the separate domain/aspect models, a sophisticated **coordination of the individual project model versions** must be maintained by the project data server at each stage of the design process. Fig. 8.22 gives an impression of the problem.

A similar problem exists at every design step.

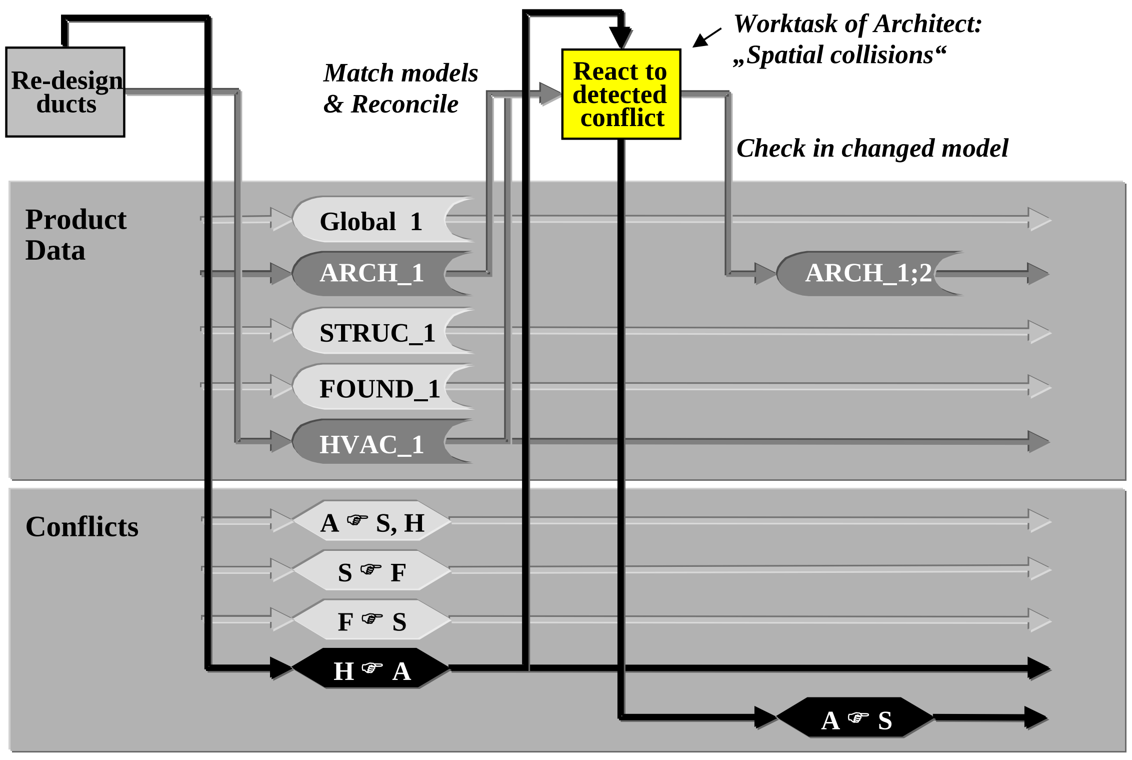


Fig. 8.22: Model data states and data conflicts before and after the work task of the architect

(1) Model data shown in light grey exist on the server, but are not used during this work task; (2) Aspect model HVAC_1 is only retrieved for comparison of changes w.r.t. the arch. model (LONG READ) (3) Aspect model ARCH_1 is checked out (LONG WRITE), compared with HVAC_1, modified and then checked in again as new model version at the end of the work task (4) conflicts shown in black are relevant to this work task, other conflicts that might exist but are not handled here are shown in light grey.

(Abbreviations: A = Architect, H = HVAC, S = Structural, F = Foundation designer)

With this final example, the integration of the developed project data and interoperability services in an extended environment for concurrent engineering, supporting not only data, but also process (workflow) management was demonstrated.

Chapter 9: Conclusions

I should say, for those who might think these things unusual, that they aren't and that they weren't difficult to find.
– Richard Mitchell, Less Than Words Can Say

In the preceding chapters, the individual components of the proposed model mapping approach for an interoperable concurrent engineering environment were separately discussed and appraised. This chapter provides a summarising evaluation of the developed concepts and the prototyped client/server implementation, and outlines open problems for future research.

9.1 Evaluation of Results

At the beginning of the thesis I suggested that a comprehensive concurrent engineering environment can only be achieved by using PDT as baseline, but extended by models, specifications and methods to support *all* information management aspects of the environment, and not only the product and process data related to the designed facilities. On the basis of this hypothesis, the main objectives of the research were formulated in section 1.2. They are fulfilled in the following way:

1. The first objective was to provide principal concepts for a distributed client/server system for concurrent engineering in building design, with emphasis on the architecture and the formal specification of interoperability models, components and services.

- Research in the area of computer integrated construction has been looking for a long time for a top-down solution that would connect the "islands of automation", and it is now largely anticipated that PDT is the most suitable technology that can glue together a CEE system comprised of a large number of different components (see e.g. Anderl 1995; Stumpf et al. 1996). However, as pointed out recently, e.g. in (Turk 1998a, b) and (Wittenoom 1998), conceptual modelling exercised without adequate concern for the actual components of a software environment, such as human activities, software applications, and related object functionality, is difficult to achieve. There are at least two problems with such "general-purpose" modelling efforts: (1) the difficulties by which the conceptual models are being formalised and agreed upon by the experts in the domain, and (2) the incompleteness of the models, particularly in the areas of unconventional and creative design (Turk 1998b).

  With the approach developed in this thesis such difficulties can be greatly overcome. Instead of looking at the problems that are directly related to the development and specification of comprehensive product and process data models, a methodology was suggested which allows to "connect" all components of a CEE system with the help of a unified modelling framework built upon already recognised model structures, such as the IFC Project Model.

This framework need not be fully harmonised as assumed by STEP. The key idea of the approach is the use of an environment-wide ontology as a specification language providing a higher level of abstraction than any specific product and process data models. Through the ontological commitment of all components of the environment, the necessary logical consistency of the framework can be guaranteed, even if individual applications have a different "understanding" of high-level concepts at their specific level of detail.

To achieve such interoperability, high-level object classes, like the abstract class *IfcRoot,* have been enhanced with behavioural features, focusing on the communication and coordination of information, and not on the engineering functionality provided in their subclasses, when treated as "normal" product data objects within the scope of a particular application. Thus, whilst there are no globally defined operations allowing e.g. to move a "wall" object, to insert a "window", and to calculate its thermal resistance or stress distribution, a set of operations is provided to check if the properties of that "wall" object have been changed by another designer, to map its representation in one domain model to another etc.

The implementation of these operations across different platforms, and by using different programming languages, is enabled by the generic representation of all information items by means of information containers, and a generic client/server communication model providing the use of different communication techniques, such as Java RMI, CORBA, CGI scripts, pure TCP/IP sockets etc. In this way, not only integration and information sharing, but also adequate consideration of many important interoperability aspects, such as data consistency and change management, can be achieved in much the same way as the simple retrieval of the attributes of some object from a database.

The developed architecture and systemic interoperability components provide an easy mechanism for the realisation of plug-ins by which applications can be integrated into the environment, the functionality of the project data server can be extended by knowledge-intensive software agents for specific fully automated tasks, and additional servers can be easily added to tackle aspects like document and workflow management, legal issues etc.

Thus, whilst there is a large number of components that can further enhance the operability of the environment, the proposed conceptual framework and software architecture can contribute to the construction of real-world CEE systems even with today's "incomplete" data models.

2. The second objective was to use PDT as baseline by aligning the developed concepts as far as possible with STEP methodology.

- All information structures supporting the underlying conceptual models of the environment (information container, communication model, mapping language, object-oriented project data operations, knowledge-based extensions to the object-oriented operations, interoperability services) are uniformly designed using as much as possible the EXPRESS modelling paradigm. Whilst not all specifications could be developed directly in EXPRESS, due to the limitations of the language, care has been taken to provide adequate links between the proposed new representations and "pure" EXPRESS data models. In this way, an environment enabling the use of STEP and/or IFC data models, along with other information sources is achieved.

In fact, some problems in the realisation of the environment can be specifically allocated to the requirement of a STEP-conformant approach. For example, the use of STEP exchange files according to (ISO 10303-21 1994), along with object-oriented requests and more advanced knowledge-based queries and assertions, provided not only implementation difficulties, but conceptual problems as well, including the unique identification of objects (impossible to represent in a STEP file if the data model does not contain the respective provisions), the needs for dual storage of the data, different treatment of conversion and filtering operations etc. Nevertheless, the undertaken study about the current state of building IT provided evidence that this is the correct approach to accomplish a real model-based environment in the medium term, as it allows to introduce advanced information processing methods without neglecting current, widely used practices.

3. The third objective was to use the IFC models as general reference models of the proposed approach by coordinating all specific components for concurrent engineering support with the IFC modelling architecture.

- The developed approach does not depend on, and is not fully aligned with the IFC Project Model. During the work on the thesis it became clear that this was not necessary for the development of most of the identified interoperability models and methods, as it was possible to define them on the level of a formal information modelling language and generic concepts common to all object-oriented models.

  However, some operations, although defined generically, had to be implemented with consideration of the specific structure of a particular data model[*)]. It was also found, as expected, that the scope of the project data services can be considerably enhanced, if the particular high-level structures of the adopted kernel model are utilised. In such cases, the IFC Project Model has been used as reference model of the envisaged environment.

  The performed examination of the IFCs showed that certain elements can be enhanced from the point of view of an IT environment for concurrent engineering. The developed approach depends on such “enhancements” as little as possible, but it also showed that by re-specifying high-level IFC constructs on the level of a system-wide shared ontology, with only a few added operational features (in this study: 4 object classes enhanced with a total of 8 operations – see appendix V) a totally new interpretation of the functionality of the IFC models can be achieved.

  This proved that the IFCs have greater potential than currently utilised, and with justifyable software adaptations it is definitely possible to envisage an IFC-based CEE system. Thus, given that IAI currently appears to be best positioned to involve the major stakeholder groups in the building industry, I see the main result of the study w.r.t. IFC in the derived optimistic conclusion about the possible usage of the IFC Project Model as a primary reference model for CEE, along with the suggested enhancements, and the proposed “interpretation” of the model in a run-time system. On that basis, the move from paper-based to model-based on-line product development is most likely to succeed.

[*)] A typical example is the generation of a drawing file from a product model.

4. The forth objective was to study and propose possibilities to incorporate advanced knowledge-based features into the concurrent engineering environment.

- As a whole, the approach suggested in this thesis is not knowledge-based. Instead, it "holds" as much as possible to the object-oriented paradigm. Even though many aspects of object-oriented modelling and programming have been criticised recently, and several insufficiencies with respect to design product modelling (cf. Garrett & Hakim 1994; Killicote et al. 1994; Sieberer & Keber 1997) and conceptual modelling in general (Borgida 1995; Russel & Norvig 1995) have been identified, the object-oriented paradigm is *intentionally* chosen as the underlying paradigm of the proposed CEE framework. There are two basic reasons for that: (1) its wide acceptance in the research and development community, and (2) its proven scalability for large information systems. More advanced approaches, such as description logic, can provide many benefits to the overall functionality of a CEE system, but they are not widely accepted and are insufficiently tested in real-world implementations. Hence, their broad applicability and their scalability could not be presumed.

  This specific design requirement put significant constraints on the use of advanced knowledge-based methods[*)]. Nevertheless, with the proposed formalisms for communication and information exchange, it was possible to implement different components of CEE by using, internally, different representation paradigms, whereas, externally, i.e. on system level, the same object-oriented interfaces based on a shared ontology could be uniformly applied. This is demonstrated by the implemented prototype environment: the project data management server was developed by using a frame-based paradigm combining object-oriented and rule-based processing, the mapping and matching methods make extensive use of symbolic programming, the front-end request broker and the example system clients are in "pure" Java, and additional components, such as a general-purpose CAD system, a structural analysis application, and a Virtual Reality Browser, are integrated on the basis of externalised, Information-Container-based requests/responses and simple object-oriented pre-processors.

  I see in this pragmatic treatment of the *Knowledge-Level* perspective (cf. Newell 1982) another major contribution of the proposed approach.

  In addition, a possibility to use advanced IT features on the project data server was found, by developing a formalism enabling the encapsulation of knowledge-based queries in "normal" object-oriented remote method calls. The specifications presented in sections 4.5 and 4.7 provide a simple mechanism to extend the capabilities of an inherently object-oriented environment. Future work can show how much more functionality can be achieved in the same generalised way.

[*)] For example, unlike KIF (Khedro et al. 1994), client applications are not assumed to possess intelligent features beyond the capabilities provided by an object-oriented representation which limits the expressiveness of client/server requests and responses.

5. The fifth objective was to develop an interoperability approach enabling the solution of problems related to the concurrent, multidiscipline work on multiple, non harmonised domain models. This objective was identified as the primary focus of the research.

- The undertaken studies showed that a common ontology can warrant high-level operability and consistency of the environment, but it is not likely to guarantee its completeness by using only the vocabulary defined in the ontology. Thus, whilst the *actions* of all components of a CEE system, including knowledge base servers and knowledge-based applications, can be coherently treated by means of their ontological commitment to high-level concepts and operations, their specific *reactions* depend on the specific interpretation of domain/application data. This data may be shared, and it may have different meaning in the context of different applications. Besides, due to the presumed organisation of project work including concurrent, autonomous, non coordinated activities, diverging model states, data conflicts and data redundancy had to be taken into account.

  In fact, the applicability of the whole approach strongly depends on how well such problems can be tackled. Input and output to/from one design domain from/to other domains raises specific model transformation problems, and parallel, autonomous processing of shared data gives rise to consistency problems which cannot be solved only by applying basic database and conceptual modelling techniques.

  In database research, due to the main interest in business DBMS, characterised by numerous, concurrent, yet short transactions, a continuous consistency of the data is always presumed, which equates semantic interoperability to schema integration. However, in chapter 5 it was shown that this is a very strong requirement w.r.t. object-oriented project models, significantly limiting the capacity of all proposed model integration methods.

  In the domain of conceptual modelling, along with the efforts for model harmonisation and the development of large consistent modelling frameworks, the need for more flexible model transformation mechanisms (model mapping) have been better recognised. However, in this domain, too, the presumption that a shared, continuously consistent data repository must be maintained has restricted most undertaken efforts to the examination of mapping/consistency problems related to the information exchange between applications and that presumed consistent repository. This strategy has been successfully proven in the limited scope of prototype environments (cf. Augenbroe 1994; Beucke 1995; Wittenoom 1998), but there are no known implementations that have succeeded in extending it to real-world systems.

  The approach developed in this thesis takes a different view on the problem domain.

  First, interoperability aspects are distinctly categorised and independently treated. This enabled the achievement of greater representational capacities of the developed specifications, as well as the coverage of a wider range of problems. The latter can be especially interesting when large applications, or separately developed large models, e.g. IFC and CIS (Crowley & Watson 2000), need to be brought together.

  Second, continuous consistency and non redundancy of the data are not imposed as leading design principles which mitigates many data conflicts, and thus enabled the tackling of complex object transformations that have not been considered before.

Third, mapping, matching, merging and reconciliation of the model data are not treated as parts of a non separable process. Instead, mapping tasks are executed without consideration of any particular already existing data context, whereas matching and merging of the data can be deferred, at the responsibility of the users of the CEE system.

The baseline of the approach is that, instead of attempting to fully automate a sophisticated information flow with a doubtful probability of success, the designers are assisted by IT tools enabling them to decide, actively, when and how the project data need to be coordinated. In this way greater modularity and flexibility can be achieved, and, what is more important, there are no strong restrictions imposed by the system w.r.t. the overall organisation of the design work. The drawback is that data integrity and data consistency cannot be guaranteed by the system alone, but require the "reasonable" behaviour of the system users. In that respect, a broad analogy to the treatment of many sophisticated CAD problems can be recognised.

In my opinion, this approach is more natural to human work, and it has greater prospects of success compared to pure model harmonisation and model integration efforts. However, as it is difficult to assess all possible consequences in situations where data conflicts cannot be fully recovered, further investigations and analyses of a large number of case studies are needed to achieve greater clarity.

6. The final specified objective was to implement a prototype software system as proof of the developed concepts.

- In chapter 8, the major components of the prototyped environment were outlined, along with a large example adapted from the final workshop of the ToCEE project. Details of the Information Container API and the prototyped project data operations were presented in appendices II and V respectively.

  The developed project data server (PROMISE) was partially used in the ToCEE project where it ran smoothly and with reasonable performance by all major tests, as well as at the final workshop demonstration. However, as only some of its features were needed in ToCEE[*)], additional debugging, testing and case studies were conducted with data of the COMBI models[**)], IFC 2.0 data, and data of the sample structure applied for the verification of the structural domain extension model developed in the diploma work (Weise 1999) mentioned in the preceding chapters. Thus, even though some functional and performance features can definitely be improved, the prototyped environment provided sufficient evidence for the overall validity and applicability of the proposed approach.

---

*) For example, there was only one "very light-weight" mapping task included in the demonstration scenarios of ToCEE, and all tasks related to the resolution of global names and actor roles were allocated to the Information Logistics System developed in the frames of the ToCEE project (Wasserfuhr & Scherer 1999).

**) These tests were needed at least for the re-construction of the COMBI examples referenced in the thesis.

9.2 Fulfilment of Basic Requirements to Design Product Models

At several places in the thesis I emphasised that the target of the developed approach and the proposed environment architecture is the design phase of building construction projects. Along with all known differences to other phases, this involves a pure IT perspective to be considered. Indeed, the information requirements of design are quite different from the information requirements of later life cycle phases, and even if the same model schemas are used, they have to be interpreted in a different way by design/construction/FM IT systems. For example, I argued before that the requirement for autonomous (private) workspaces is of leading importance, taking precedence over consistency and integration issues, and influencing the overall concept of the CEE system. However, whilst I believe that this is true for a design environment, it does not necessarily apply to later phases where the information content is greater but less “vulnerable”, as there are much less changes and alternatives that need to be tackled. In such cases, the use of a shared project database is more justified and easier to implement than in design. Furthermore, in contrast to construction and facilities management, where an (almost) fully instantiated building model exists on input, in design the input is comprised mostly of requirements which, at best, only sparsely “fill” the model schemas, whereas the instantiated building model is in fact the result of the design activities.

Because of these, and other related aspects, design product models are subject to more complex requirements w.r.t. IT compared to the models of later life cycle phases. However, intensive examinations of such specific design requirements performed by many researchers in the last years (cf. Sriram 1991; Eastman 1993; Hakim 1993; Björk 1995; Kowalczyk 1997) have shown that most of them are not related to the model structure and content but to the representation and manipulation of the data in a running system. Some of these requirements, i.e. data integrity, data redundancy, flexibility and extensibility of the models, uniformity of the representation, were already addressed in the previous section. Other major requirements that are often mentioned include schema and object evolution, representation and tackling of complex entities, derived data, user-defined semantics, generation of views, versioning, independence of the data storage from applications, persistence. An assessment of how well these requirements are fulfilled by the developed prototype project data server (PROMISE) is provided below.

Schema evolution

PROMISE can accommodate several models at a time, and all generic server modules, i.e. more than 80% of the program code, can be used with different EXPRESS-based modelling frameworks. This capability was manifested through the performed studies with models from COMBI, ToCEE, and IFC 2.0 + modifications (see chapters 7, 8 and appendix VII). Therefore, in principle, the representation of evolving model schemas provides no problems. Locally, this can be done at any level and includes creation/deletion/modification of object classes, class relationships, value types, cardinalities, even inheritance types*). However, at the level of client/server communication, schema evolution is intentionally limited because of several integrity and consistency concerns. PROMISE was not conceived for model development, but for run-time support of multidiscipline design teams where such excessive flexibility of the representation may be a dangerous feature.

*) In fact, most of these features are provided by the basic capabilities of the KEE system itself (see Intellicorp 1994).

Object evolution

Object evolution can be interpreted in two ways:

(1) as a subtopic of schema evolution, and

(2) as the ability to support partial instantiation, by enabling the assignment of features defined in the object classes not only when respective object instances are first constructed, but also dynamically, according to the specific output of design activities.

Whilst the first of these aspects is quite arguable for a run-time CEE system, the second is an inherent feature of design that must be supported without ifs and buts.

In PROMISE this second aspect of object evolution is provided in the following way (see chapter 8 and appendix V):

Object instantiation is supported by a “create” operation which automatically constructs a new unique object instance in the referenced model, and fills only those attributes that are provided as parameters in the respective request. Locally, all object instances can be modified at any time as long as they comply with the model schema known to the server. Such modifications become globally visible once the checked out data are checked in again. It is also possible to redefine operations as long as their signatures remain unchanged. In addition, capabilities for object re-classification are provided, but these are less developed than e.g. in description logic, and require additional rules to be stored with the model schemas. Objects can also be instantiated partially or fully due to mapping operations. In that case, the content of each target model instance depends on the content of the respective source instance(s). Subsequent matching and merging operations may be used to complete the properties of such objects.

Complex entities

If complex entities are represented as aggregations in the underlying data models, they can be queried and retrieved both as a whole and as individual "part-of" components. This can be done for all parts, or by using a filtering predicate, such as "all elements with $h > 0.3\,m$".

If aggregations are missing, ad hoc groups can be created and retrieved by means of the proposed knowledge-based expressions and templates (see section 4.7)

Derived data

In EXPRESS data models, derived data appear only in the form of derived attributes. Such attributes can be populated easily by internal demon methods. This is done automatically, after the values of the independent attributes are known to the system.

User-defined semantics

User-defined semantics can be accommodated in each domain model. Because of the relative independence of domain models from each other, extended semantic relationships and/or operations need only be known to the users and applications “using” a particular domain model, and are “shadowed” w.r.t. all other system components (see chapter 4).

However, there are certainly some “limits of flexibility” w.r.t. the overall consistency and integrity of the data. The precise specification of such limits is a difficult problem and has not been attempted in this study.

Views

In addition to domain/application models different views can easily be obtained by built-in operations or ad hoc constructed knowledge-based queries (see chapter 4 and appendix V). However, some of the built-in operations for views, e.g. geometry view types, are strongly dependent on the internal model structures and might require changes if the underlying model is modified (see section 9.1, item 3).

Versioning

Currently, PROMISE supports only model-level versions, and an “undo” feature enabling to rollback the latest transaction (see chapter 4 and appendix V). More advanced version support methods have been developed in other research projects, e.g. REMAP. However, according to the findings of REMAP (cf. Sieberer & Keber 1997), in order to provide comprehensive version support additional representational features have to be incorporated in the model schemas. The combination of such features with the data structures used in PROMISE is an interesting topic for future research.

Independence of the data storage from the use of the data by applications

This aspect of design product models is an inherent feature of the proposed framework.

Operations applied to design entities by applications are always performed in the context of their discipline-specific view of the project data, whereas storage, shared access and coordination of the data are controlled by overarching interoperability methods executed on the project data server. In addition, the invocation of each remote operation is always checked against the assigned access rights in accordance with the roles of the users of the CEE system (see chapter 4 and appendix V).

Persistence

All models maintained by the system are persistently stored, including not only the current, but also previous and/or temporary versions, resulting e.g. from mapping, matching or merging operations.

Model and mapping schemas are stored together with the specific operations defined for and in these schemas. This feature allows to separate model-dependent operations from the globally valid generic operations defined at the system ontology level.

However, knowledge-based queries are only possible when the whole referenced model is loaded into working memory, which restricts the scalability of the approach. Therefore, in order to provide such advanced server features in real-world CEE implementations, further research work aiming at combining knowledge-base and database technologies is needed, as suggested in contemporary database research (cf. Ullman 1988; Eastman et al. 1995a; Kim 1995; Schönhoff et al. 1997).

9.3 Directions for Future Research

Information technology subjects related to concurrent engineering provide a vast area for research. Almost any aspect of the proposed framework for CEE can be further developed, and a variety of sophisticated data management tools that would improve project co-ordination, monitoring and control can be envisaged. Even when only the personal workplace of a single designer is considered, a number of tasks that are not efficiently supported by today's building IT can easily be identified, e.g. simulation of the behaviour of the building due to different environmental influences (design for operability), simulation of construction problems (design for assemblability), forecasting and pre-emptive measures for probable life cycle situations (design for serviceability), activity planning and coordination of the designer's work across several projects (design as information flow) etc. This list can be readily extended to fill chapters of a book.

The aim of this final section is more modest. It focuses on ideas for future research that are closely related to the proposed approach for a model-based interoperable concurrent engineering environment described in this thesis. These ideas can be broadly divided into five categories: (1) investigation of advanced interoperability issues, (2) extensions of the modelling framework, (3) extensions of the principal functionality of the CEE system, (4) extensions of PROMISE, and (5) extensions related to the user interfaces.

Investigation of advanced interoperability issues

In the discussions provided in chapters 4-7 several interoperability aspects that need further examination were already mentioned, and in section 9.2 the need for a more formal identification of the limits of model modifications was addressed.

A basic research topic related to semantic interoperability is the completeness of the mapping and integration approaches. It is probably not possible to develop methods providing complete coverage of all inter-schema problems associated with object-oriented model schemas due to the incompleteness of the underlying logic model of the object-oriented paradigm, but it is certainly interesting to investigate possibilities for predicting if a mapping transformation can be applied successfully. Formal methods, allowing to estimate in advance if the changes made in one domain model will be adequately propagated to other domain models, can be very useful for the proper coordination of design decisions and can help in optimising information flows. More pragmatic, short-term efforts may address flexible visualisations of changes, identification of critical sections of the model data with great overlaps of discipline specific information, more comprehensive examination of relationship paths, i.e. the influence of the changes in one modelling object to other objects in the model etc.

Another area that requires attention is the development of appropriate modelling tools to support the development and analysis of mapping specifications. While a large number of tools for creating, browsing and analysing of object-oriented models are currently available, similar tools for the development of mapping specifications are practically missing. However, when large conceptual models are involved, such tools can be of tremendous importance to the developers of a CEE system. In that respect, the mapping patterns presented in chapters 5 and 6 may provide some initial hints. Additional work on the extension of these patterns can be combined with graphical user interfaces allowing to define visually inter-schema correspondences on selected appropriate levels of detail. More precise investigations of mapping patterns can also be useful for early estimations of the difficulties and the expected run-time performance of mapping tasks.

Further ideas w.r.t. interoperability include:

- development of computationally more efficient algorithms;
- better interactive control of the mapping, matching and merging processes;
- negotiation methods related to data conflicts;
- incorporation of more advanced knowledge-based features and more intelligent communication mechanisms;
- better alignment with standards supported on the Internet, such as XML, etc.

Extensions of the modelling framework

In the approach developed in this thesis, the problems related to the requirements, the specification and the quality of conceptual project models were not directly addressed. However, a CEE system is certainly strongly dependent not only on the structuring of the data and the realisation of the interoperability aspects of the underlying modelling framework, but also on its content. There are numerous technical issues in the domain of conceptual modelling for which adequate solutions are still needed, including e.g. design rationale, integrity control on well-defined data levels, flexible extensions to the model schemas provided by the users at the time of the creation of instances, better user and application interfaces etc.

Five years ago Björk (1995) predicted that when product model based applications start to be used in real construction projects, a number of interesting research topics will emerge, such as:

- further studies of information use and information flows;
- the influence of the use of product and process models on the design process;
- integration of product model research with design theory.

With the appearance of IFC-based applications these issues gain even more importance in the present day.

Another topic for further research related to the modelling framework of CEE is the ontological level. In the scope of this work, only some initial specifications w.r.t. the system-wide ontological commitment to a kernel project model were developed. The defined operations, comprising a mixture of primitive SDAI-like data access functions and higher-order functions for interoperability support, are intended as building blocks for more comprehensive project data services that could be used not only implicitly, through appropriate IT tools, but also explicitly, for direct user/server communication via the Internet. This feature needs further investigations to prove its validity for real-world design environments involving a variety of legacy applications. Further work is needed also for more exact specifications of such project data services.

An interesting subject for research can be also the definition of an “engineering language” that would enable not only the programmatic implementation of the ontology specifications but their direct use with WWW-Browsers and PDM-Browsers. It should be possible to find a formalism by which end-users would be able to “speak” with the CEE system using words like “column” and beam”, and not “IfcBuildingElement”, “IfcColumn”, “IfcBeam”, “IfcRelAssignsTypedProperties” and so on. Obvious but insufficient solutions include the construction of dedicated viewers and the creation of synonym tables. I suppose that AI methods related to natural language analysis need to be examined here as well.

Extensions of the principal functionality of the CEE system

Whilst I argued that model-based project realisation is imperative for efficient practising of concurrent engineering in building design, there are also many other forms of information related to the design process that have to be taken into account. This includes the representation and management of design documentation, design processes, legal issues, such as contracts, access rights and responsibilities, etc. Of course, these types of data are also based on conceptual model schemas, and it is principally possible, and necessary, to integrate them in the overall modelling framework of CEE. However, the information content of these schemas and the required services are very different from the data types examined in object-oriented project models such as IFC. Therefore, it is probably not sufficient to define appropriate "plug-in" elements in each model and hope that this will do the job. More detailed investigations are needed to address not only the data aspects and the functionality of the related systems (PDM, EDM, WfM …), which is currently as far as research efforts have gone (see e.g. Fisher & Froese 1996; Scherer 2000), but also the interoperability services and the inter-dependencies between these different model worlds.

With respect to workflow management it is interesting to examine e.g. the use of project model data as input, output and execution conditions for worktasks, in place of the traditionally practised association of documents to activities. Some work in this area already exists (cf. Wasserfuhr & Scherer 1997, 1999), but there is still much research needed to achieve practical applicability of such large-scale approaches.

Other aspects that can extend the functionality of a CEE system include the integration of a legal framework for the virtual enterprise, as suggested e.g. in (Scherer 1997b), standards and regulation processing (cf. Killicote et al. 1995; Turk 1996) etc.

Extensions of PROMISE

Although the focus of this work has not been on implementation issues, the developed prototype project data server could be used successfully in various contexts. This allowed to identify several possible extensions for future work:

1) Implementation of agent technology to extend the basic services of PROMISE by supporting client applications in the solution of tasks beyond their specific domains (e.g. a knowledge-based structural design system could “use” a server agent to prove if the provisions of some design standard are fulfilled w.r.t. certain modelling objects).
2) Implementation of distributed database technology to allow decentralised storage of the project data.
3) Linking to external product information systems to enable flexible selection and integration of supplier products in the design.
4) Incorporation of comprehensive conflict management functionality.
5) Realisation of comprehensive version control on finer level.
6) Improvement of the interoperability methods, along with future investigations of mapping patterns and consistency issues, etc.

Another possible direction of work is the examination of other representation paradigms and the associated re-engineering of the system. Here there are two contrasting possibilities to explore: (1) advanced representation paradigms, such as description logic, which can lead to extended functionality but are uncertain w.r.t. scalability, and (2) pure object-oriented representations based on a widely accepted programming language with comprehensive Internet capacities, such as C++ or Java. However, especially in the second case, several principles of the software design might need to be reconsidered.

Extensions related to the user workplace

The personal workplace of the designer has not been a prime issue of interest in this thesis. Work in this direction was limited to the implementation of example clients and the demonstration of a few design applications. However, for the success of a CEE system, the workplace and the tools of the individual designers are of no less importance than the framework and the operability of the overall system. Some hints in that respect were given in the preceding paragraphs, e.g. the use of WWW-Browsers for model access, more sophisticated interfaces enabling end-users to recognise better (and faster) the changes made to the design data etc.

A specific issue closely related to conceptual modelling is the development of user-friendly project data browsers. Currently, the capabilities of academic and commercially developed browsers are strongly aligned with the structures of the represented models. This is good for model developers but inconvenient for end-users. A designer will not be interested in tracing long chains of objects, as for example "*IfcWall – IfcMaterialLayerSetUsage – IfcMaterialLayer – IfcMaterial – IfcSimpleProperty – IfcMeasureValue – …*", in order to gather, piece by piece, the chunks of information he needs. He would rather like to select the "wall" object visually, and leave the system to collect and show the necessary data according to the criteria he has specified. However, this requires a "knowledge overlay" to the conceptual model schemas, which is not contained in current project data models. In that respect, too, there is much work yet to be done.

⌘

Alles Wissen und alle Vermehrung unseres Wissens endet nicht mit einem Schlusspunkt, sondern mit einem Fragezeichen.

– Hermann Hesse, Lektüre für Minuten

Appendices

This part includes seven appendices intended to facilitate the reader in understanding better the given examples and referenced external sources at different places in the preceding body of text. Each of these appendices presents supplementary technical details to the discussed topics in chapters 1-9. More comprehensive descriptions can be found in the cited literature sources.

Appendix I provides a brief overview of EXPRESS and EXPRESS-G.

Appendix II details the prototyped Java API for the Information Container specification introduced in section 4.2.

Appendix III presents the XML DTD suggested in section 4.8 as an alternative syntax for the externalisation of the Information Container data structures and the client/server communication based upon that.

Appendix IV presents the developed XML DTD for CSML externalisation mentioned in the discussion at the end of section 6.7.

Appendix V presents the prototyped project data management operations used in the validation of the developed concepts.

Appendix VI contains a brief description of the prototyped parsers and converters for the various proposed representation formats.

Finally, **Appendix VII** provides an overview of the data models referenced at several places throughout this study. It includes five subsections, covering, respectively:

1) the STEP modelling framework;
2) the IFC modelling framework;
3) the COMBI modelling framework;
4) the ToCEE modelling framework, and
5) EXPRESS-G diagrams of selected data model schemas used in the prototyped project data server for CEE (the IFC Kernel Model, the proposed structural domain model extension to IFC 2.0 and a small application-specific model used for testing purposes with the latter).

Appendix I Overview of EXPRESS/EXPRESS-G

This appendix presents a brief overview of the EXPRESS modelling language and its graphical subset, EXPRESS-G, compiled from different literature sources (Fowler 1995; Owen 1997; IAI 1999c). It is provided for information and to assist readers in the interpretation of various modelling concepts and examples illustrated with the help of EXPRESS and EXPRESS-G in this thesis. A complete reference of the capabilities of the language is provided in (ISO 10303-11 1994).

I.1 EXPRESS constructs

EXPRESS is a textual data definition language. It is based on an extended Entity-Relationship modelling paradigm, includes generalisation and constraint specifications, and embodies many characteristics of object-oriented modelling.

One of the key aspects of EXPRESS is that it is both "computer-interpretable" and "human-readable". It conforms to a formal syntax, so that models can be validated and processed by software, but can also be presented to a human reader in a form that allows a data specification to be readily understood. This latter aspect of EXPRESS is supported by its graphical subset, EXPRESS-G.

In the context of STEP, EXPRESS is designed for *implementation independent conceptual product modelling*. However, its use is not limited to product data models and to STEP. In fact, EXPRESS has been widely used in other standardisation work, e.g. EDIFACT, and in many industrial, research and academic projects. Together with UML, it is the favourite presentation format for reported work in the conceptual modelling domain today.

The basic constructs and modelling capabilities of EXPRESS are as follows:

<u>Schema</u>

A *schema* represents the highest specification level in EXPRESS. Every conceptual model consists of one or more schemas, each defining a common scope for a collection of data definitions. Inter-schema interfacing allows different components of large models to be developed separately. However, there is no provision for public and private specifications as in object-oriented programming and other data definition languages, i.e. all defined concepts in a model are visible globally, in all its schemas.

<u>Data types</u>

EXPRESS supports the use of the following data types in a schema specification:

- base data types,
- declared data types, and
- entities.

The *base types* are `BINARY`, `NUMBER`, `INTEGER`, `REAL`, `BOOLEAN`, `LOGICAL` and `STRING`. They correspond more or less to the usual data types in programming languages.

A `NUMBER` is a generalisation of `INTEGER` and `REAL`, and `LOGICAL` is a specialisation of `BOOLEAN` (extending the available Boolean values `TRUE` and `FALSE` with the additional value `UNKNOWN`). There are no specific `INTEGER` and `REAL` types for different scopes and precision, such as *short*, *long*, *double*, *float*, found in other languages.

Declared data types are used to create additional data types with their own unique identifiers. There exist two kinds of declared data types in EXPRESS: *constructed data types* (`ENUMERATION` and `SELECT`), and *defined data types*.

An `ENUMERATION` data type provides for a range of possible values specified in an enumeration list. An item of that type may only take one value from the given range of values. For example, a 'space' type that may be selected from an enumerated list of 'occupied', 'technical' or 'circulation' can be specified as a data type as follows[*)]:

```
TYPE IfcSpaceTypeEnum = ENUMERATION OF
                           (Occupied, Technical, Circulation);
END_TYPE;
```

A `SELECT` data type defines a named collection of other data types that can play the same role in a model. There need not be anything in common between the types declared in a select list. As with enumerations, only one item of a select list can be used as a value of the `SELECT`. For example:

```
TYPE IfcBuildingSelect = SELECT
                           (IfcBuilding, IfcBuildingStorey);
END_TYPE;
```

declares a data type that can be used to reference both 'building' and 'building storey' objects, but only one at a time.

A *defined data type* is used to take the place of another, underlying data type that is already known in the schema. Typically, this is a base type that is given a more appropriate name in this way. For example, an 'organisation' may have a short description which could take the form of a simple `STRING`, but it would be more expressive to refer to that description as 'label'; this is achieved by the definition:

```
TYPE label = STRING;
END_TYPE;
```

Entities

Entities are the basic conceptual modelling units within EXPRESS. In many aspects they resemble classes in object-oriented programming languages. Concepts like generalisation and specialisation with multiple inheritance, as well as value and reference attributes and behaviour are supported. Thus, each entity declaration *creates a class* and *gives it a name*. For example:

```
ENTITY IfcOwnerID;
  Identifier : IfcString;
  OwningApp  : IfcString;
  OwningUser : IfcActor;
END_ENTITY;
```

Here, `Identifier`, `OwningApp` and `OwningUser` are all *attributes* of the entity `IfcOwnerID`, the first two being of a defined data type with underlying base type (`STRING`), and the last being a reference attribute pointing to another entity (`IfcActor`).

EXPRESS is unusual by comparison to many other data modelling languages in that it does not make a distinction between attributes and relationships. In EXPRESS, an attribute is

---

[*)] All EXPRESS examples in this appendix are adapted from the IFC 2.0 documentation (IAI 1999c).

regarded as defining the role played by a base type or a defined type in the definition of an entity; a relationship is similarly regarded as the role played by one entity type in the definition of another.

Subtypes and supertypes

Generalisation/specialisation relationships between the entity types in a data model are provided in the following way.

If one entity type is defined to be a subtype of another, then it inherits all its properties, i.e. its definition, attributes and constraints. Multiple inheritance is supported, where one entity type is defined as a subtype of two or more other entity types. The resulting classification lattice enables the definition of complex entity instances, combining the characteristics of several “parent” types. When an entity type has more than one defined subtype, the default (somewhat unusual) relationship, denoted as `ANDOR`, is that the subtypes may be instantiated independently or together. This may be constrained using a `SUPERTYPE` clause in the definition of the parent entities. In fact, constraining inheritance relationships to single inheritance only is one of several additional modelling rules imposed on the IFC modelling framework. For example, consider the following definitions:

```
ENTITY IfcLayeredElement
  ABSTRACT SUPERTYPE OF
    (ONEOF (IfcFloor, IfcRoofSlab, IfcWall));
  …
END_ENTITY;

ENTITY IfcWall
  SUBTYPE OF (IfcLayeredElement);
  …
END_ENTITY;
```

Here, by using the keyword `ONEOF`, the *supertype* declares that a 'layered element' is exclusively either a 'wall', or a 'floor', or a 'roof slab'; it cannot be two or more at the same time. The 'layered element' itself cannot be instantiated as it is declared `ABSTRACT`.

Attributes

As mentioned, EXPRESS does not make distinction between value and reference attributes, but it does recognise different possible cardinalities and constraints. Thus, an attribute may be mandatory or optional, it may have a unique or a derived value, and it may stand in one to one, or one to many relationship with the defining entity class.

A mandatory attribute is specified simply through a reference to the attribute’s type, and for optional attributes the keyword `OPTIONAL` is provided to denote that the presence or absence of the respective attribute does not affect the basic meaning of an entity instance.

EXPRESS provides `UNIQUE` and `DERIVE` rules for cases, where the specific values of one attribute, or a combination of attributes, are required to be unique across the full set of their values in a given model, and for cases, where the value of an attribute can be calculated from the value of other attributes, or by applying a predefined expression or function, respectively.

Except for simple one-to-one relationships between entities and their attributes, a number of *aggregation relations* are supported as well. These include: `ARRAY` – a fixed size collection of data items of the same type with order; `BAG` – a collection of data items with no order and allowed duplications, `SET` – a collection of items with no order and no duplication, and `LIST` – an ordered sequence of items, optionally declared as `UNIQUE`, i.e. with no duplications. For each of these aggregate types the lower and upper bounds of the aggregation may be specified. In bags, lists and sets, a zero (`0`) specified as the lower bound denotes that the aggregation is optional (may contain zero elements), and a question mark (`?`) specified as upper bound denotes that the maximal number of elements is not fixed.

Inverse attributes

The representation of relationships between entities in EXPRESS appears to be one-directional. In fact, this is not the case: *all* relationships are essentially bi-directional, but EXPRESS makes only the "more important half" explicit. However, there is always an implicit reverse relationship. If necessary, this can be captured explicitly by means of `INVERSE` rules. An `INVERSE` rule does not create new relationships, it just makes the reverse relationship visible by giving it a name, and allows to constrain it and to specify the needed multiplicity (optional, one to one, multiple bag, set, list values etc.).

Local constraints

In addition to `UNIQUE`, `DERIVE` and `INVERSE`, it is possible to specify certain constraints that apply to every instance of an entity class. Such local constraints are introduced within the definition of the entity with the help of `WHERE` rules. Each `WHERE` rule is a logical expression which must evaluate to `TRUE` to satisfy the specified requirement for the given attribute value(s).

The two (almost complete) IFC entities in the example below expose most of the discussed features of EXPRESS attributes. `IfcCovering` illustrates the specification of required, optional, derived and inverse attributes, and a local constraint. `IfcApplication` shows how strings can be constrained to have fixed maximum number of characters, as well as two unique rules – for a single attribute, and for a group of two attributes.

```
ENTITY IfcCovering
  SUBTYPE OF (IfcBuildingElement);
    PredefinedType   : IfcCoveringTypeEnum;
    LayerInformation : IfcMaterialLayerSetUsage;
    calcCoveringArea : OPTIONAL IfcAreaMeasure;
  DERIVE
    SELF\IfcBuildingElement.HasMaterial :
         IfcMaterialSelect := LayerInformation.ForLayerSet;
  INVERSE
    AttachedTo : SET [0:?] OF IfcRelAttachesToBoundaries
                 FOR RelatedCoverings;
  WHERE
    WR62: 'IFCPROPERTYRESOURCE.IFCMATERIALLAYERSET' IN
                 TYPEOF(SELF\IfcBuildingElement.HasMaterial);
END_ENTITY;

ENTITY IfcApplication;
    ApplicationIdentifier : STRING(16);
    ApplicationFullName   : STRING(255);
    Version               : STRING(255);
  UNIQUE
    UR1: ApplicationIdentifier;
    UR2: ApplicationFullName, Version;
END_ENTITY;
```

Note that in `IfcCovering` the derived attribute, `HasMaterial`, presents a specialisation of the same attribute in one of the ancestors of the entity, restricting it only to values "derived" from the given expression. The attribute `IfcRelAttachesToBoundaries` represents the inverse relationship to `RelatedCoverings` which is also specified in one of the entity's ancestors.

Other features

In addition to the presented "basics" of EXPRESS, there exist many other features and constructs that contribute to the representational power of the language. These include: algorithmic units (`FUNCTION`, `PROCEDURE`, `RULE`), standard constants, functions and procedures, global constraints, schema interfacing (`USE`, `REFERENCE`) etc.

The complexity of the language may be construed from the fact that its reference manual (ISO 10303-11 1994) is more than 200 pages long, and the formal language specification includes 122 different keywords and 318 syntax productions (Fowler 1995).

I.2 EXPRESS-G constructs

EXPRESS-G is a graphical notation for a *subset* of EXPRESS intended for human communication. Everything drawn in EXPRESS-G can be defined in EXPRESS, but not everything that can be defined in EXPRESS can be drawn in EXPRESS-G. However, although it does not support all the features of the language, EXPRESS-G provides for cross-referencing between schemas and for multi-page diagrams for a single schema. This is fortunate, as EXPRESS-G is a generally "verbose" language, and requires more diagrams for a given model than other graphical languages, such as ER, NIAM or UML.

In particular, EXPRESS-G supports:

– SCHEMA level diagrams, enabling to illustrate schemas and inter-schema links, and

– ENTITY level diagrams, enabling the graphical specification of references to definitions in other schemas, multi-page references, `ENTITY` definitions, `TYPE` definitions, attributes, relationships and cardinalities, as well as several "shortcuts" for the presence of constraints.

Expressions, functions and other "more verbose" textual elements cannot be represented in EXPRESS-G. `ARRAYS`, `BAGS`, `SETS` and `LISTS` are abbreviated to **`A`**, **`B`**, **`S`** and **`L`** respectively. Other "shortcuts" in relationship names include: **`(DER)`** for derived, **`(INV)`** for inverse and **`(RT)`** for redefined attributes, as well as asterisks (**`*`**) denoting the presence of a `WHERE` and/or `UNIQUE` rule.

Fig. I.1 on the next page presents most of the available graphical symbols in EXPRESS-G. Appendix VII provides the full graphical notation of the IFC Kernel Model for reference, and the full graphical notation of a model proposed as a structural domain extension to the IFC framework. Spanning over 7 pages and referencing at many places the "upper layers" of the IFC framework, this domain model can be useful also as an illustrative example for the capabilities provided by the EXPRESS-G notation.

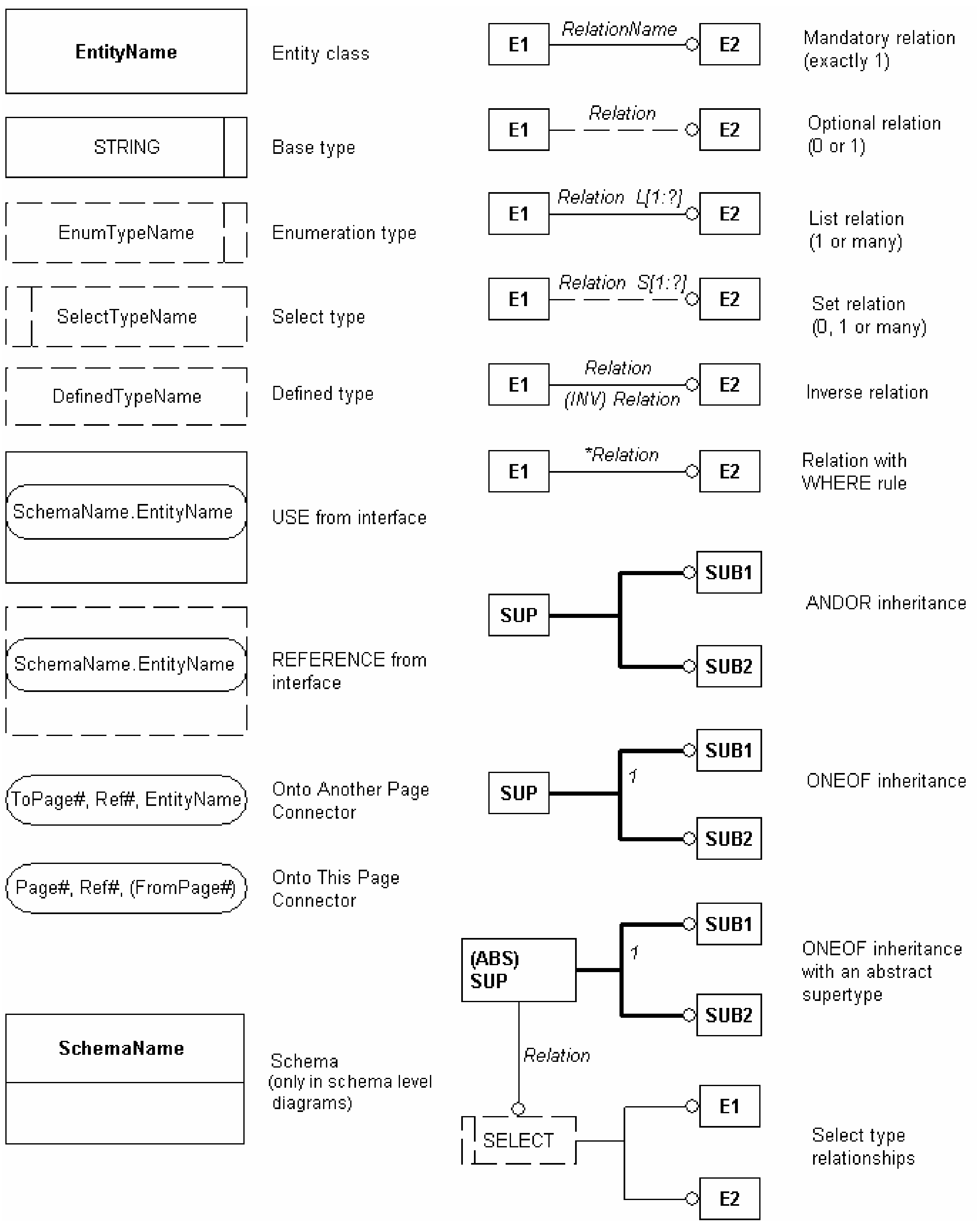


Fig. I.1: EXPRESS-G graphical symbols

Appendix II Information Container API for Java

The formal Information Container specification presented in section 4.2 has been designed so that an implementation of its constructs in an object-oriented programming language can be easily accomplished.

As an example, this section presents the developed Information Container API for Java, i.e. the available classes and public methods that can be used directly by client applications written in Java version 1.1 or higher (see e.g. Heller et al. 1997; Kühnel 1997; JDK 1.1 or SDK 1.2 at java.sun.com etc.).

According to their functionality the classes defined in the Information Container library (package `ptdms.ic`) can be subdivided into three categories: *public classes*, *exception classes* and *utility classes*.

The ***public classes*** provide straight-forward mapping of the components of the Information Container specification. Each of them contains several methods enabling their use by applications written in Java. They all have one or more *constructor methods* that allow to create respective instances by providing their "external" (textual) representation as input. Symmetrically, all they possess a method `toString()` which returns syntactically correct externalisations of the respective object instances as string values. In addition, a *Parser* class is provided to support the definition of Information Container subclasses if needed for specific applications.

The ***exception classes*** enable differentiated treatment of errors that are associated with the syntax and semantics of the Information Container specification.

The ***utility classes*** are used internally for RMI-based communication. They are not public and may not be invoked directly by an application. Therefore, these classes are not shown in the following description of the Information Container API.

II.1 Class hierarchy

Primary classes:

class java.lang.Object
- class ptdms.ic.**InfoContainer** (implements java.io.Serializable)
- class ptdms.ic.**Aggregation** (implements java.io.Serializable)
- class ptdms.ic.**Select** (implements java.io.Serializable)
- class ptdms.ic.**ObjectRef** (implements java.io.Serializable)
- class ptdms.ic.**BLOB** (implements java.io.Serializable)
- class ptdms.ic.**ICSymbol** (implements java.io.Serializable)
  - class ptdms.ic.**ICSname** (implements java.io.Serializable)
  - class ptdms.ic.**ICLogical** (implements java.io.Serializable)
- class ptdms.ic.**ICParse**r
- class ptdms.**CEEsession** (implements java.io.Serializable) *)

*) This class is actually not a part of the Information Container package. It is included here for convenience, as it is used in several methods of the API.

Exception classes:

class java.lang.Object
    class java.lang.Throwable (implements java.io.Serializable)
        class java.lang.Exception
            class ptdms.ic.**FeatureTypeException**
            class ptdms.ic.**MalformedICstringException**
            class ptdms.ic.**MalformedICsyntaxException**

Utility classes (for internal use):

class java.lang.Object
    class java.rmi.server.RemoteObject
        class java.rmi.server.RemoteServer
            class java.rmi.server.UnicastRemoteObject
                class ptdms.ic.adapters.**ICRmiImpl** (implements ic.ptdms.adapters.ICRmiInter)
interface java.rmi.Remote
    interface ptdms.ic.adapters.**ICRmiInter**

II.2 Class definitions

This section presents a detailed specification of the **public object classes** of the Information Container API. A description of the utility classes is not provided as these classes are intended only for internal use.

The constructors and public methods of each class are given in tabular form, with full specification of the method signature, a short functional description and, where applicable, a list of possible exceptions. For conciseness, the exceptions thrown by the separate object methods are abbreviated as follows:

FType	=	FeatureTypeException
MString	=	MalformedICstringException
MSyntax	=	MalformedICsyntaxException

For standard Java exceptions the following abbreviations are used:

NullPtr	=	NullPointerException
IllegalArg	=	IllegalArgumentException
ArrayOut	=	ArrayOutOfBoundsException
IO	=	IOException
MURL	=	MalformedURLException
Security	=	SecurityException

A **FeatureTypeException** is thrown when an attempt is made to read or write a feature with a wrong data type.

A **MalformedICstringException** is thrown when an incorrect token is encountered in the externalisation of an Information Container or one of its component.

A **MalformedICsyntaxException** is thrown when the externalisation of an Information Container or one of its components does not match the defined EBNF syntax.

Standard Java exceptions have their usual meaning.

Class ptdms.ic.InfoContainer

```
java.lang.Object
|
+----ptdms.ic.InfoContainer
```

Synopsis:

This is the main class of the Information Container API. It is used to package and represent *all* types of data that can be exchanged in client-server communication on the basis of the developed project data management services. For reference, the EBNF syntax of the *InfoContainer* externalisation, as defined in section 4.2, is as follows:

```
InfoContainer      =  label '(' { feature }* ')' .
```

where:

```
label              =  symbol | refID .
feature            =  symbol ':' valueSelect .
valueSelect        =  typedValueSelect | InfoContainer .
```

Class Declaration:

`public class **InfoContainer** extends Object implements Serializable`

Constructors:

Constructor signature	Description	Throws
InfoContainer( )	The default empty constructor.	
InfoContainer(CEEsession *sess*)	Creates an empty *InfoContainer* instance and assigns it a session.	Security
InfoContainer(String *s*, boolean *XMLinput*) **InfoContainer**(CEEsession *sess*, String *s*, boolean *XMLinput*)	Creates an *InfoContainer* instance from string *s*, which must be a valid InfoContainer externalisation. If *sess* is provided, the InfoContainer is assigned to a session, and if *XMLinput* is present and *true*, the input string *s* is expected to conform to the alternative syntax defined by the XML DTD presented in App. III.	MString Msyntax Security[*)]
InfoContainer(String *lbl*, String[ ] *names*, Object[ ] *values*) **InfoContainer**(CEEsession *sess*, String *lbl*, String[ ] *names*, Object[ ] *values*)	Creates an *InfoContainer* instance with label *lbl* containing the features (name-value pairs) specified in the two parameter lists of the constructor call. If *sess* is provided, the InfoContainer is assigned also to the respective session.	IllegalArg Ftype Security[*)]

Public Methods:

Method signature	Description	Throws
Aggregation **getAggregAt**(String *f*)	Retrieves the *Aggregation* object stored at feature *f*.	FType
BLOB **getBLOBAt**(String *f*)	Retrieves the *BLOB* object stored at feature *f*.	FType

[*)] Only for the second form of the constructor.

boolean **getBooleanAt**(String *f*)	Retrieves the *boolean* value stored at feature *f*.	FType
Class **getClassAt**(String *f*)	Retrieves the class of the object stored at feature *f*.	
String[ ] **getFeatureNames**( )	Returns in a String array all feature names contained in an InfoContainer.	
Hashtable **getFeatureTable**( )	Returns all features contained in an InfoContainer as a reference to a *Hash-table*, with keys – the feature names, and values – the feature values (as *Objects*).	
Class[ ] **getFeatureTypes**( )	Returns an array of "class" objects for all features contained in an InfoContainer.	
Object[ ] **getFeatureValues**( )	Returns an array of objects for all feature values contained in an InfoContainer.	
InfoContainer **getICAt**(String *f*)	Retrieves the *InfoContainer* object stored at feature *f*.	FType
String **getLabel**( )	Returns the label of this InfoContainer.	
ICLogical **getLogicalAt**(String *f*)	Retrieves the *logical* value (instance of ICLogical) stored at feature *f*.	FType
long **getLongAt**(String *f*)	Retrieves the *long* value stored at *f*.	FType
ObjectRef **getObjectRefAt**(String *f*)	Retrieves the *ObjectRef* object stored at feature *f*.	FType
double **getRealAt**(String *f*)	Retrieves the *double* value stored at *f*.	FType
Select **getSelectAt**(String *f*)	Retrieves the *Select* object stored at feature *f*.	FType
CEEsession **getSession**( )	Returns the session in which this InfoContainer is created, or *null* if there is no assigned session.	
String **getStringAt**(String *f*)	Retrieves the *String* stored at feature *f*.	FType
ICSymbol **getSymbolAt**(String *f*)	Retrieves the symbol object (*ICSymbol*) stored at feature *f*.	FType
Object **getValueAt**(String *f*)	Generic access method to retrieve any type of object stored at feature *f*.	
boolean **isAggregAt**(String *f*)	Tests if the value of feature *f* is an aggre-gation.	
boolean **isBLOBAt**(String *f*)	Tests if the value of feature *f* is a BLOB, i.e. a reference to an external data file to be uploaded or downloaded, e.g. a STEP physical file.	
boolean **isBooleanAt**(String *f*)	Tests if the value of feature *f* is a *boolean* value.	
boolean **isFeature**(String *f*)	Tests if feature *f* exists in this InfoContainer instance.	

boolean **isICAt**(String *f*)	Tests if the value of feature *f* is a valid InfoContainer instance.	
boolean **isLogicalAt**(String *f*)	Tests if the value of feature *f* is a boolean value, i.e. *true*, *false* or *unknown*..	
boolean **isLongAt**(String *f*)	Tests if the value of feature *f* is a long integer number.	
boolean **isObjectRefAt**(String *f*)	Tests if the value of feature *f* is an object reference, i.e. instance of *ObjectRef*.	
boolean **isRealAt**(String *f*)	Tests if the value of feature *f* is a real number (mapped to Java *double*).	
boolean **isSelectAt**(String *f*)	Tests if the value of feature *f* is an instance of the *Select* object class.	
boolean **isStringAt**(String *f*)	Tests if the value of feature *f* is a *String*.	
boolean **isSymbolAt**(String *f*)	Tests if the value of feature *f* is a symbol, i.e. a predefined enumeration item, a class or an attribute name.	
int **numberOfFeatures**( )	Returns the number of contained features.	
String **setLabel**(String *lbl*)	Sets the InfoContainer label.	NullPtr
Object **setValueAt**(String *f*, Object *obj*)	Sets a new value for feature *f*, or adds the feature with the given value to the InfoContainer if it is a new feature.	NullPtr FType
String **toString**( )	Overrides *java.lang.Object.toString( )* providing a syntactically correct exter-nalisation of InfoContainer objects. By using *toString( )* it is possible to “pipe” the output of a server response from Java to another implementation written in some other programming language, which provides its own InfoContainer parser on the basis of the specified syntax*).	
String **toXMLString**( )	Provides a syntactically correct exter-nalisation of InfoContainer objects according to the XML DTD presented in Appendix III.	
Void **XMLOutput** (OutputStream out, String encoding, String indent)	Writes a XML InfoContainer document to the OutputStream *out*. The argument *encoding* provides the name of the encoding to use, UTF-16 or ISO-8859-1. If not null, the argument *indent* provides the desired indentention for “pretty printing”.	IO NullPtr

*) Another possibility to connect the Java Information Container API to an application written in another programming language is to provide *wrapper classes* for each component class of the API. Such wrapper classes are responsible to translate the InfoContainer components into some meta format, more suitable for processing by that other language. In fact this technique has been used also for the project data server which is implemented in Common LISP.

Class ptdms.ic.Aggregation

```
java.lang.Object
|
+----ptdms.ic.Aggregation
```

Synopsis:

This class is used to represent aggregations like lists, sets, bags and arrays (as defined in ISO 10303-11 1994). The EBNF syntax of the legal externalisation of *Aggregation* objects is:

```
 aggregation       =  '[' { valueSelect }* ']' .
```

where:

```
 valueSelect       =  typedValueSelect | InfoContainer .
```

Class Declaration:

```
public class Aggregation extends Object implements Serializable
```

Constructors:

Constructor signature	Description	Throws
Aggregation( )	Default empty constructor.	
Aggregation(Vector *v*)	Constructs an *Aggregation* instance and fills it with the elements of Vector *v*.	NullPtr FType
Aggregation(String *agg*)	Creates an *Aggregation* instance from string *agg*, which must be a valid aggregation (list or set) externalisation.	MString MSyntax FType

Public Methods:

Method signature	Description	Throws
void **add**(Object *obj*)	Appends a new element at the end of an aggregation.	NullPtr IllegalArg
Aggregation **getAggregAt** (int *index*)	Retrieves the *Aggregation* instance found at position *index*.	FType ArrayOut
BLOB **getBLOBAt**(int *index*)	Retrieves the *BLOB* instance found at position *index*.	FType ArrayOut
boolean **getBooleanAt**(int *index*)	Retrieves the *Boolean* instance found at position *index* and converts it to the basic *boolean* type.	FType ArrayOut
Object **getElementAt**(int *index*)	Retrieves the *Object* found at pos. *index*.	ArrayOut
InfoContainer **getICAt**(int *index*)	Retrieves the *InfoContainer* instance found at position *index*.	FType ArrayOut
ICLogical **getLogicalAt**(int *index*)	Retrieves the *ICLogical* instance found at position *index*.	FType ArrayOut
long **getLongAt**(int *index*)	Retrieves the *Long* instance found at position *index* and converts it to the basic *long* type.	FType ArrayOut

ObjectRef **getObjectRefAt** (int *index*)	Retrieves the object reference (*ObjRef*) found at position *index*.	FType ArrayOut
double **getRealAt**(int *index*)	Retrieves the *Double* instance found at position *index* and converts it to the basic *double* type.	FType ArrayOut
Select **getSelectAt**(int *index*)	Retrieves the *Select* instance found at position *index*.	FType ArrayOut
int **getSize**( )	Returns the number of elements in an aggregation.	
String **getStringAt**(int *index*)	Retrieves the *String* instance found at position *index*.	FType ArrayOut
ICSymbol **getSymbolAt**(int *index*)	Retrieves the *ICSymbol* instance found at position *index*.	FType ArrayOut
Object **getValueAt**(int *index*)	Retrieves the value stored at pos. *index*.	ArrayOut
Vector **getVector**( )	Returns the content of an aggregation instance as a Java *Vector*.	
void **insert**(Object *obj*, int *index*)	Inserts a new element at position *index* in an aggregation.	ArrayOut IllegalArg
boolean **isAggregAt**(int *index*)	Tests if the value at *index* is an aggregation.	ArrayOut
boolean **isBLOBAt**(int *index*)	Tests if the value at *index* is a *BLOB* object.	ArrayOut
boolean **isBooleanAt**(int *index*)	Tests if the value at *index* is a *boolean*.	ArrayOut
boolean **isICAt**(int *index*)	Tests if the value at *index* is an *InfoContainer*.	ArrayOut
boolean **isLogicalAt**(int *index*)	Tests if the value at *index* is an *ICLogical* object.	ArrayOut
boolean **isLongAt**(int *index*)	Tests if the value at *index* is a long integer number.	ArrayOut
boolean **isObjectRefAt**(int *index*)	Tests if the value at *index* is an instance of the *ObjectRef* class.	ArrayOut
boolean **isRealAt**(int *index*)	Tests if the value at *index* is a real (mapped to *double* in the Java API).	ArrayOut
boolean **isSelectAt**(int *index*)	Tests if the value at *index* is a *Select*.	ArrayOut
boolean **isStringAt**(int *index*)	Tests if the value at *index* is a *String*.	ArrayOut
boolean **isSymbolAt**(int *index*)	Tests if the value at *index* is a symbol i.e. an *ICsymbol* object.	ArrayOut
void **remove**(int *index*)	Removes the element at *index*.	ArrayOut
void **replace**(Object *obj*, int *index*)	Replaces the element at position *index* with the object *obj*.	ArrayOut IllegalArg
String **toString**( )	Overrides *java.lang.Object.toString( )* providing a syntactically correct externalisation of an aggregation.	

Class ptdms.ic.Select

```
java.lang.Object
|
+----ptdms.ic.Select
```

Synopsis:

This class represents `SELECT` data types as defined in (ISO 10303-11 1994). The EBNF syntax of the legal externalisation of *Select* objects is:

```
 select           =  symbol '(' selectValueType ')' .
```

where:

```
 selectValueType  =  literal | Obj_Ref | structuredValueType .
```

Class Declaration:

```
public class Select extends Object implements Serializable
```

Constructors:

Constructor signature	Description	Throws
Select( )	Default empty constructor.	
Select(ICSymbol *name*, Object *obj*) **Select**(ICSymbol *name*, boolean *b*) **Select**(ICSymbol *name*, long *l*) **Select**(ICSymbol *name*, double *d*) **Select**(ICSymbol *name*, String *s*) **Select**(ICSymbol *name*, ICLogical *logicalValue*) **Select**(ICSymbol *name*, Aggregation *agg*) **Select**(ICSymbol *name*, ObjectRef *oRef*) **Select**(ICSymbol *name*, Select *sel*)	Constructs a *Select* instance with the specified name and value. The alternative forms of the constructor allow to store different types of values in a *Select*, without the need of an explicit type casting.	NullPtr
Select(String *selstr*)	Creates a *Select* instance from the string *selstr*, which must be a valid select type externalisation.	MString MSyntax

Public Methods:

Method signature	Description	Throws
boolean **getAggregation**( )	Retrieves the *Aggregation* object stored in this *Select* instance.	FType
boolean **getBoolean**( )	Retrieves the *boolean* value stored in this *Select* instance.	FType
boolean **getClass**( )	Retrieves the *Class* of the underlying data type.	
ICLogical **getLogical**( )	Retrieves the *ICLogical* object stored in this *Select* instance.	FType

long **getLong**( )	Retrieves the *long* value stored in this *Select* instance.	FType
ICSymbol **getName**( )	Retrieves the name of the select type.	
double **getReal**( )	Retrieves the *double* value stored in this *Select* instance.	FType
ObjectRef **getObjectRef**( )	Retrieves the *ObjectRef* stored in this *Select* instance.	FType
Select **getSelect**( )	Retrieves the *Select* stored in this *Select* instance (in the case of a nested select type).	FType
String **getString**( )	Retrieves the *String* value stored in this *Select* instance.	FType
String **getSymbol**( )	Retrieves the *ICSymbol* object stored in this *Select* instance.	FType
Object **getValue**( )	Retrieves the value stored in this *Select* instance as a generalised *Object* instance.	
boolean **equals**(Object *obj*)	Overrides *java.lang.Object.equals( )*. This method enables the correct comparison of *Select* objects.	
boolean **isAggregation**( )	Tests if the value of the *Select* is an *Aggregation* object.	
boolean **isBoolean**( )	Tests if the value of the *Select* is a *boolean* value.	
boolean **isLogical**( )	Tests if the value of the *Select* is an *ICLogical* object.	
boolean **isLong**( )	Tests if the value of the *Select* is a long integer number.	
boolean **isReal**( )	Tests if the value of the *Select* is a real (mapped to *double* in the Java API).	
boolean **isObjectRef**( )	Tests if the value of the *Select* is an instance of the *ObjectRef* class.	
boolean **isSelect**( )	Tests if the value of the *Select* is itself a *Select*.	
boolean **isString**( )	Tests if the value of the *Select* is a *String*.	
boolean **isSymbol**( )	Tests if the value of the *Select* is an *ICSymbol* object.	
String **setName**(String *name*) String **setName**(ICSymbol *name*)	Sets the name of this *Select* instance.	NullPtr MSyntax
Object **setValue**(Object *obj*)	Sets the value of this *Select* instance.	NullPtr MSyntax
String **toString**( )	Overrides *java.lang.object.toString( )* providing a syntactically correct externalisation of *Select* objects.	

Class ptdms.ic.ObjectRef

```
java.lang.Object
|
+----ptdms.ic.ObjectRef
```

Synopsis:

This class represents remote object references in the Information Container API. It enables a client application to address and execute methods of any registered remote object on the project data server.

The EBNF syntax of the legal externalisation of *ObjectRef* is:

```
Obj_Ref  = OREF '(' refID ')' .
```

where:

```
refID    = modelID | objID .
modelID  = ID .
objID    = [ modelID ':' ] ID .
ID       = symbol [ '.' { symbol | longint } [ version ] ] .
```

Class Declaration:

```
public class ObjectRef extends Object
                       implements Serializable
```

Constructors:

Constructor signature	Description	Throws
ObjectRef( )	Default empty constructor.	
ObjectRef(CEEsession *sess*)	Creates an empty *ObjectRef* instance and assigns it a session.	Security
ObjectRef(String *oref*)	Creates an *ObjectRef* instance from the string *oref*, which must be a valid object reference externalisation.	MString MSyntax
ObjectRef(CEEsession *sess*, String *oref*)	Same as above, but assigns also a session.	MString MSyntax Security

Public Methods:

Method signature	Description	Throws
InfoContainer **execMeth** (String *meth*, InfoContainer *par*) InfoContainer **execMeth** (String *meth*, InfoContainer *par*, boolean *SyncMode*)	This method establishes automatically an appropriate connection with the project data server and executes the remote method *meth* for the referenced object, returning the result provided by this method packed as a new *InfoContainer* instance. The parameter *par* packages all the parameters for the remote method (when the method requires no parameters, the value of *par* should be set to *null*).	MURL MString MSyntax Security

ObjectRef **getClassRef**( )	Retrieves the class of the object reference as an addressable object reference itself.	
String **getClassByName**( )	Returns the class name of the referenced object as a *String* suitable for use as an 'OID' parameter in a client request.	
ObjectRef **getConcept**( )	Synonymous method to *getClassRef( ).*	

ObjectRef **getContent**( )	Returns the content of the *ObjectRef* instance as an *InfoContainer* object.
	
	This method is in fact a more efficient and straight-forward implementation of:
	
	execMeth("inspect", null, true)
	.getICAt("content")
	.getICAt("attributes")
	(see section V.1 for a description of the "inspect" operation).
		MURL
		Security

String **getObjectID**( )	Retrieves the unique ID of the referenced object in the project.	
CEESession **getSession**( )	Returns the session in which this object reference is established, or *null* if there is no assigned session.	
String **setObjectID** (String *id*)	Sets the object ID to *id*. This method is legal only for the server maintaining the unique IDs of the respectively registered object classes.	NullPtr Security
String **toString**( )	Overrides *java.lang.Object.toString( )* providing a syntactically correct externalisation of an *ObjectRef*.	

Class ptdms.ic.BLOB

```
java.lang.Object
|
+----ptdms.ic.BLOB
```

Synopsis:

This class is used to represent binary large objects (BLOBs). It allows a client application to upload or download BLOBs, i.e. local files on the client's computer, to/from the project data management server. Such files can be in any of the acceptable formats by the server, currently STEP physical file format for upload and download, and DXF, VRML, HTML and ASCII text for download only[*)]. Normally, BLOB would be used in conjunction with remote method invocation for ObjectRef instances, where the input or output parameters are specified as BLOBs, typically for creating or checking in/out product data contained in STEP physical files.

The EBNF syntax of the externalisation of BLOB is:

```
BLOB_Ref = BLOB '(' refID ')' .
```

where:

```
refID    = modelID | objID .
modelID  = ID .
objID    = [ modelID ':' ] ID .
ID       = symbol [ '.' { symbol | longint } [ version ] ] .
```

Class Declaration:

```
public class BLOB extends Object implements Serializable
```

Constructors:

Constructor signature	Description	Throws
BLOB( )	Default empty constructor.	
BLOB(CEEsession *sess*)	Creates an empty *BLOB* instance and assigns it a session.	Security
BLOB(String *blob*)	Creates a *BLOB* instance from the string *blob* which must be a valid BLOB exter-nalisation.	MString MSyntax
BLOB(CEEsession *sess*, String *blob*)	Same as above, but assigns also a session.	MString MSyntax Security

*) VRML and DXF output is primarily intended for viewing/presentation purposes. It is achieved with the help of a generically defined server method. However, the implementation of this method is not independent of the specific product data model as it needs concrete knowledge of the way geometry is represented in the model. In the prototype implementation of the project data server this method is realised for the COMBI, ToCEE and IFC data models, the latter two only for 'Bounding Box' geometric representations.

Public Methods:

Method signature	Description	Throws
String **getBlobID**( )	Retrieves the BLOB ID.	
InputStream **getInStream**( )	Creates an input stream and associates it with the BLOB for subsequent upload. A similar public method for respective downloading of BLOBs, such as e.g. *getOutStream( )* is not needed, since the output stream is created automatically by the server. Applications are not allowed (and not required) to influence the download process.	IO
CEEsession **getSession**( )	Returns the session in which this BLOB instance is created, or *null* if there is no assigned session.	
void **loadFromFile**(String *fname*)	Uploads a local file to the server and associates it with the BLOB object. The user or application must have appropriate access rights to execute this method.	IO Security
void **saveAsFile**(String *fname*)	Downloads the file associated with the BLOB from the server and saves it locally in the specified file. As with *loadFromFile(…)*, the user or application must have appropriate access rights to execute this method.	IO Security
void **setBlobID**(String *id*)	Sets the BLOB ID. The execution of this method also depends on the actual access rights of the user or the application.	NullPtr Security
void **setInStream**(InputStream *is*)	Sets the input stream associated with the BLOB object and prepares the upload of the data. A similar public method for downloading of BLOBs, such as e.g. *setOutStream( )* is not needed, since an output stream is created automatically by the server.	IO
String **toString**( )	Overrides *java.lang.Object.toString( )* providing a syntactically correct exter-nalisation of *BLOB* objects.	
URL **toURL**( )	Converts the BLOB reference into a URL object (useful for http-based download/upload).	

Class ptdms.ic.ICSymbol

```
java.lang.Object
|
+----ptdms.ic.ICSymbol
```

Synopsis:

The ICSymbol class is used basically for representing class names, attribute names and enumeration values contained in Information Containers. There is a constructor provided for this class, but it is seldom necessary to create ICSymbol instances explicitly as they are generated automatically when needed by the InfoContainer constructor itself.

The EBNF syntax of the legal externalisation of a symbol is:

```
symbol  = letter { letter | digit | '_' }* .
```

Class Declaration:

```
public class ICSymbol extends Object implements Serializable
```

Constructors:

Constructor signature	Description	Throws
ICSymbol( )	Default empty constructor.	
ICSymbol(String *sym*)	Creates an *ICSymbol* instance from the string provided in the parameter *sym*.	IllegalArg

Public Methods:

Method signature	Description	Throws
boolean **equals**(Object *obj*)	Overrides *java.lang.Object.equals( )*. This method can be used to compare *symbol* objects correctly.	
String **toString**( )	Overrides *java.lang.Object.toString( )*. This provides a syntactically correct externalisation of *ICSymbol* and its subclasses *ICSName* and *ICLogical* in the Information Container API.	
String **valueOf**( )	Returns the string representation of an *ICSymbol*. This method is in fact identical to the method *toString( )* and is provided only for conceptual reasons, to enable the handling of *symbol* objects in the same way as the wrapper objects for the literals *long*, *double*, *boolean* etc., which all contain a respective *valueOf( )* method.	

Class ptdms.ic.ICSName

```
java.lang.Object
|
+----ptdms.ic.ICSymbol
     |
     +----ptdms.ic.ICSName
```

Synopsis:

This object class is used primarily to enable the construction of *refIDs* and InfoContainer *labels*. Unlike symbols, *refIDs* and *labels* may contain special characters, such as `'.'`, `':'`, `';'` and `'-'`. In any other aspects *ICSname* is identical to *ICSymbol*.
ICSName objects are also created automatically by the InfoContainer constructors whenever necessary, the class' own constructors are seldom needed.

Class Declaration:

`public class` **`ICSName`** `extends ICSymbol implements Serializable`

Constructors:

Constructor signature	Description	Throws
ICSName( )	Default empty constructor.	
ICSName(String *name*)	The primary *ICSName* constructor. Creates an *ICSName* instance from the string provided in the parameter *name*.	IllegalArg

Public Methods:

Method signature	Description	Throws
boolean **equals**(Object *obj*)	Overrides *ptdms.ic.ICSymbol.equals( )* providing correct comparison for *labels* and *refID*s.	
String **toString**( )	Inherited from *ptdms.ic.ICSymbol*.	
String **valueOf**( )	Inherited from *ptdms.ic.ICSymbol*.	

Class ptdms.ic.ICLogical

```
java.lang.Object
|
+----ptdms.ic.ICSymbol
      |
      +----ptdms.ic.ICLogical
```

Synopsis:

This class represents `LOGICAL` data types as defined in (ISO 10303-11 1994). It is treated similar to enumerations, except that the enumeration values are always constant symbols, i.e. *true*, *false* or *unknown*.

Class Declaration:

`public class` **`ICLogical`** `extends ICSymbol implements Serializable`

Constructors:

Constructor signature	Description	Throws
ICLogical( )	Default empty constructor.	
ICLogical(String *str*)	Creates an *ICLogical* instance from the string provided in the parameter *str*.	IllegalArg
ICLogical(boolean *booleanValue*)	Creates an *ICLogical* instance from the *boolean* value provided by the parameter *booleanValue*. This constructor cannot be used to create an *ICLogical* object containing the value *unknown*.	

Public Methods:

Method signature	Description	Throws
boolean **getBoolean**( )	If the logical value stored in this instance is *true* or *false*, it is returned as a boolean value. If the value is unknown, an exception is thrown.	FType
boolean **equals**(Object *obj*)	Overrides *ptdms.ic.ICSymbol.equals*( ) providing correct comparison of *logical* values.	
boolean **isBoolean**( )	Checks if the value of this logical object can be converted to *boolean*.	
String **toString**( )	Inherited from *ptdms.ic.ICSymbol*.	
String **valueOf**( )	Inherited from *ptdms.ic.ICSymbol*.	

Class ptdms.ic.ICParser

```
java.lang.Object
|
+----ptdms.ic.ICParser
```

Synopsis:

This is a convenience class that can be used to parse the external representation of an InfoContainer *before* creating it. It contains a set of public class methods useful for tokenizing an InfoContainer string and for querying the type or content of a token.

Class Declaration:

`public final class` **`ICParser`** `extends Object`

Constructors:

none.

Public Methods:

Method signature	Description	Throws
boolean **parse**(String *icstr,* boolean *XMLinput*)	Checks if *icstr* is a valid InfoContainer externalisation. Unlike the respective constructor of the *InfoContainer* class, *parse* does not throw an exception, but simply returns *false* when an error occurs. If *XMLinput* is *true, icstr* is expected to conform to the alternative syntax defined by the XML DTD presented in Appendix III.	
boolean **testToken**(Object *testObj,* Class *class*)	Tests if an InfoContainer token (*testObj*) is an instance of a given class.	
boolean **testToken**(Object *testObj,* Object *pObj*)	Tests if an InfoContainer token (*testObj*) is equal to the pattern object *pObj*.	
Vector **tokenizeICstring**(String *s*)	Tokenizes an InfoContainer string into component constructs according to the InfoContainer specification, i.e. *ObjRef, symbol, aggregation* etc.	MString MSyntax

Class ptdms.CEEsession

```
java.lang.Object
|
+----ptdms.CEEsession
```

Synopsis:

CEEsession is used to obtain a session from the project data server. A session is primarily needed to dispatch properly client requests in asynchronous communication mode, where all responses can be packaged and routed together to the client. Within a session a local variable closure is established which allows the execution of several ‘linked’ requests. CEEsession is not part of the **`ptdms.ic`** package. Its brief description is included here for convenience.

Class Declaration:

`public class` **`CEEsession`** `extends Object`

Constructors:

Constructor signature	Description	Throws
CEEsession( )	The default session constructor. Establishes a new session and returns a respective *CEEsession* object for use in *InfoContainer*, *Aggregation*, *ObjectRef* and *BLOB* object methods.	Security

Public methods:

Method signature	Description	Throws
InfoContainer **execRequest** (ObjectRef *reqRef*) InfoContainer **execRequest** (String *reqStr*)	Executes a request created in and associated with this session (the details of the *Request* class are not presented here for conciseness).	
InfoContainer **getClient**( )	Returns the available information about the client that has issued the session.	
InfoContainer **getHost**( )	Returns the available information about the host of the client in the session.	
ICSymbol[ ] **getProtocols**( )	Returns the applicable communication protocols, e.g. TCP, RMI, HTTP.	
boolean **isProtocol**(String *prot*) boolean **isProtocol**(ICSymbol *prot*)	Tests if the comm. method provided by *prot* is available for this session.	

II.3 Example use of the Information Container API for Java

The following Java function retrieves the parameters of an *IfcBoundingBox* instance describing the geometry of an *IfcBeam*, provided that a bounding box representation is found among the geometric representations of the *IfcBeam* instance (see IAI 1999c). An *ObjectRef* for this *IfcBeam* is assumed to be available at the server's data repository.

```
import ptdms.*;
import ptdms.ic.*;
import java.util.Vector;
import java.rmi.*;
Vector BboxForElement(ObjectRef oref) throws Exception {
// oref is the referenced entity (i.e. the IfcBeam object)
InfoContainer ic;            // local variables for temporary
Aggregation   aggr, aggr2;   // Information Container structures
ObjectRef     shapeRef;
Vector bbox = new Vector();  // this Vector accomodates the result
try {
  // get the content of the IfcBeam instance
  // with the help of the server operation "inspect";
  ic = oref.execMeth("inspect",null).getICAt("attributes");
  // check if "Representations" exist
  if (ic.isFeature("Representations") &&
      ic.isAggregAt("Representations")) {
      // get and scan the list of all available Representations
      aggr = ic.getAggregAt("Representations");
      for (int i=0; i<aggr.getSize(), i++) {
        // get the content of the referenced Representation
        ic = aggr.getObjectRefAt(i).execMeth("inspect",null);
        ic = ic.getICAt("attributes");
        // check if "ShapeRepresentations" exist
        if (ic.isFeature("ShapeRepresentations") &&
          ic.isAggregAt("ShapeRepresentations")) {
          // scan the list of all ShapeRepresentations
          aggr2 = ic.getAggregAt("ShapeRepresentations");
          for (int j=0; j<aggr2.getSize(); j++) {
            shapeRef = aggr2.getObjectRefAt();
            // check if it is an "IfcBoundingBox" instance
            if (shapeRef.getClassByName().equals("IfcBoundingBox")){
               // get the content and store it in Vector bbox
               ic = shapeRef.execMeth("inspect",null);
               ic = ic.getICAt("attributes");
               bbox.add(ic.getElementAt("Corner"));
               bbox.add(ic.getElementAt("Xdim"));
               bbox.add(ic.getElementAt("Ydim"));
               bbox.add(ic.getElementAt("Zdim"));
               // as only 1 BoundingBox per elem. is allowed
               // return immediately the extracted values
               return bbox;
          } }  // end inner if block and inner for loop
      } }      // end outer if block and outer for loop
  } else throw new Exception("No bounding box repr. found.");
} catch (Exception e) { /* more error checking if needed ... */ }
} // end BboxForElement(ObjectRef oref).
```

Comment:

The above function will work properly by RMI-based communication between the client application and the project data server, i.e. when the distributed object architecture of Java is used. However, by TCP/IP-based communication, in place of **`execMeth(…)`** called three times in the example code (lines 14, 22, 34), the generic request format providing the InfoContainer content in externalised form, as specified in section 4.2, should be used instead.

To accommodate both approaches, **`execMeth(…)`** should be overloaded in a subclass of *ObjectRef*, e.g. as follows:

```
InfoContainer execMeth (String meth, InfoContainer par, boolean syncMode,
                        CEESession sess)
                        throws Exception {
  InfoContainer ic = null;     // initialisation
  if (sess.isProtocol("RMI"))  // test the communication protocol
     // the current comm. protocol is RMI, therefore:
     // call execMeth and return the result (an InfoContainer)
     return execMeth(meth, par);
  else if (sess.isProtocol("TCP")) {
     // the current comm. protocol is TCP/IP, therefore:
     // execute a (string-based) TCP/IP request which returns
     // a response with the result of the called method contained
     // in the output parameter "content", also an InfoContainer
     ic = sess.execRequest("request(OID:\"" + getObjectAddress() + "\"" +
                           "meth:" + meth + "inParams:" + par.toString() +
                           "syncMode:" + syncMode + ")" );
     if (ic.getSymbolAt("status").valueOf().equals("ok"))
        return ic.getFeatureAt("content");
     else
        throw new Exception(ic.getStringAt("exceptionMessage"));
  }  // end else if block
  else
     // not a known protocol, throw an Exception
     throw new Exception("Non valid communication mode");
}
```

The second approach, presented by the outer *else-if* block, is obviously less elegant, but on the other hand, it allows to implement the given method in a front-end pre-processor for an application written in another language, and not using the Information Container API. In fact, with this approach the application does not have to bother about the communication with the server at all; this task can be entirely delegated to the Java pre-processor.

The chained invocation of **`execMeth(…)`** and **`getICAt(…)`** in line (14) is used to retrieve in one step the attributes of an object instance packaged in an InfoContainer by "**`inspect`**" (see section V.1 further below for a description of the **`inspect`** operation).

Appendix III XML DTD for Information Container Externalisation

Due to their clear, hierarchical structuring, the components of the InfoContainer model can easily be mapped to a corresponding XML-based representation[*)]. This alternative syntax, enabled by the well-formed semantics of the developed model, has certain advantages as follows:

– XML data is easily parsable. There exist two well-defined standardised parsing methods: an event-driven method (SAX), and a model-based method (DOM). A number of software implementations are available and can be adopted for parsing a XML-based Information Container externalisation.

– XML provides a non restrictive representation format which makes existing applications less vulnerable to future extensions of an existing DTD.

– The processing of Information Containers with WWW-Browsers can be realised more efficiently on the basis of XML which facilitates *http-based* communication.

– At last, as XML is becoming a de facto standard for the exchange of structured data, its use can contribute to the wider acceptance of the suggested modelling concepts.

However, this approach has also some disadvantages.

First, the adaptation of Information Containers to XML leads to an inevitable design compromise due to the fact that XML has actually been designed for the structured representation of documents whereas Information Containers are intended basically for the representation of object-oriented product data.

Second, the XML format is quite verbose. Thus, the use of Information Containers is less compact than with the dedicated syntax introduced in section 4.2 which has negative consequences especially when greater amounts of data need to be exchanged.

Third, an implementation of the Information Container API on the basis of the DOM model, enabling the automated parsing of Information Container data with off-the-shelf parsers, would lead to complex object-oriented data structures that are more difficult (and unnatural) to use.

Last but not least, the data representation in XML is less explicit compared to the syntax presented in section 4.2. It does not cover the full range of data types needed which leads to much more sophisticated type resolution methods. As a consequence, a pre-processor for XML-based Information Container data requires greater internal knowledge of the underlying data models which greatly reduces the advantages of using a standardised parser.

Therefore, XML is suggested basically as an **alternative externalisation format for WWW-Browser based clients**, and *not* as a replacement to the Information Container syntax introduced in chapter 4 and the Information Container API detailed in Appendix II. The two utility methods *toXMLString* and *XMLOutput* of the *InfoContainer* class in the Information Container API provide the necessary functionality for this approach.

[*)] There exists plenty of literature about XML. A good overview of the features, the syntax, the scope and the areas of application of XML is provided e.g. in (Goldfarb & Prescod 1998).

III.1 Formal specification of the XML DTD for Information Container Externalisation

The following DTD presents the externalisation of the Information Container constructs in XML format. All Information Container specific data types (*InfoContainer*, *Feature*, *Aggregation*, *Object Reference*, *BLOB*, *Select*, *Symbol*, *Expression*), as well as the basic data types in EXPRESS, are provided as XML elements with respectively defined tags. Most of these tags are chosen so that the InfoContainer externalisation does not become too verbose. Thus, *InfoContainer* is mapped to **`<IC>`**, *Feature* to **`<F>`** (short form), *Aggregation* to **`<AGG>`**, *Object Reference* to **`<OREF>`**, *BLOB* to **`<BLOB>`**, *Symbol* to **`<SYM>`**. The basic EXPRESS data types are provided as follows: BOOLEAN is mapped to **`<BOOL>`**, LOGICAL to **`<LOGICAL>`**, INTEGER to **`<INT>`**, REAL to **`<REAL>`** and STRING to **`<STR>`**. In addition, ENUMERATION is mapped to **`<ENUM>`** and SELECT to **`<SEL>`**. LIST, BAG and SET all map to **`<AGG>`**, and NUMBER is mapped to **`<INT>`** or **`<REAL>`** depending on the value.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--  Information Container DTD Specification for
      XML-based communication and exchange of AEC data.
-->
<!-- Base element: may contain 1 or more InfoContainers
     although currently only one InfoContainer is supported
     by the implemented InfoContainer API.
-->
<!ELEMENT XIC (IC)+ >
<!--
  Entity definitions
-->
<!ENTITY % label "LBL CDATA #REQUIRED" >
<!ENTITY % featureElement "FEATURE | F" >
<!ENTITY % icValue "IC" >
<!ENTITY % typedValueSelect
           "BOOL|LOGICAL|INT|REAL|STR|ENUM|EXPR|OREF|BLOB|AGG|SEL" >
<!ENTITY % selectValueType
           "BOOL|LOGICAL|INT|REAL|STR|ENUM|OREF|AGG|SEL" >
<!ENTITY % valueSelect  "%icValue; | %typedValueSelect;" >
<!ENTITY % booleanValue "VAL (true | false) #REQUIRED" >
<!ENTITY % logicalValue "VAL (true | false | unknown) #REQUIRED" >
<!ENTITY % integerValue "VAL CDATA #REQUIRED" >
<!ENTITY % realValue    "VAL CDATA #REQUIRED" >
<!ENTITY % stringValue  "VAL CDATA #REQUIRED" >
<!ENTITY % enumValue    "VAL NMTOKEN #REQUIRED" >
<!ENTITY % expression   "VAL CDATA #REQUIRED" >
<!ENTITY % symbolName   "NAME NMTOKEN #REQUIRED" >
<!ENTITY % refID "REFID CDATA #REQUIRED" >
<!-- elementID and elementRef are provided for future extension:
     to enable the definition of more than one InfoContainer in
     a XML document, as well as references b/n the elements within
     that document.
-->
<!ENTITY % icElementID  "ID ID #IMPLIED" >
<!ENTITY % icElementRef "HREF IDREF #IMPLIED" >
```

```
<!--
  Elements
-->
<!-- The basic InfoContainer data type:
     contains a list of 0 or more features -->
<!ELEMENT IC (%featureElement;)* >
<!ATTLIST IC %label; >
<!-- Feature: name-value pair corresponding to EXPRESS attributes.
     The name is given as a XML attribute, the value can be one of
     several data types providing a mapping for the allowed EXPRESS
     attributes in a data model. -->
<!ELEMENT FEATURE (%valueSelect;) >
<!ATTLIST FEATURE %symbolName; >
<!-- Shortcut for FEATURE -->
<!ELEMENT F (%valueSelect;) >
<!ATTLIST F %symbolName; >
<!-- Simple data types -->
<!ELEMENT BOOL EMPTY >
<!ATTLIST BOOL %booleanValue; >
<!ELEMENT LOGICAL EMPTY >
<!ATTLIST LOGICAL %logicalValue; >
<!ELEMENT INT EMPTY >
<!ATTLIST INT %integerValue; >
<!ELEMENT REAL EMPTY >
<!ATTLIST REAL %realValue; >
<!ELEMENT STR EMPTY >
<!ATTLIST STR %stringValue; >
<!-- ENUM allows the mapping of EXPRESS enumeration items
     to be recognised correctly; except for its use context,
     it is semantically equivalent to other symbolic names
     represented by the entity %symbolName in this DTD -->
<!ELEMENT ENUM EMPTY >
<!ATTLIST ENUM %enumValue; >
<!-- EXPR can be used to define embedded knowledge-based
     operations enabling enhanced data queries -->
<!ELEMENT EXPR EMPTY >
<!ATTLIST EXPR %expression; >
<!-- ObjectReference: enables remote references to EXPRESS
     entities (classes or instances) -->
<!ELEMENT OREF EMPTY >
<!ATTLIST OREF %refID; >
<!-- BLOB: enables the specification of large objects as elements
     of a XML document, typically ISO 10303-21 files -->
<!ELEMENT BLOB EMPTY >
<!ATTLIST BLOB
          URL CDATA #IMPLIED
          MIME-TYPE NMTOKEN "SPF"
>
```

```
<!-- Structured data types -->
<!-- Aggregation: generalized type representing EXPRESS Lists,
     Sets, Bags and Arrays -->
<!ELEMENT AGG (%valueSelect;)* >
<!ATTLIST AGG TYPE NMTOKEN #IMPLIED >
<!-- SELECT: represents EXPRESS SELECT constructs -->
<!ELEMENT SEL (%selectValueType;) >
<!ATTLIST SEL %symbolName; >
<!-- End of the InfoContainer DTD Specification -->
```

III.2 Examples

To provide a better basis for comparison, the two examples shown in section 4.2.3 are re-used here. They illustrate many of the mentioned advantages and disadvantages of the XML approach. In addition, a third example adapted from the first example in section 4.3.4 demonstrates how a request/response sequence can be recorded with the help of XML, enabling the display of session logs by XML-aware WWW-Browsers.

Example 1: *(see section 4.2.3, page 76)*

The Information Containers shown in the original example, i.e.:

```
1. line.L1(pnt:oRef(P1) dir:oRef(V1))
2. cartesian_point.P1(coordinates:[0.0 0.0 0.0])
3. vector.V1(orientation:oRef(D1) magnitude:2.5)
4. direction.D1(direction_ratios:[1.0 1.0 0.0])
```

can be represented in the following way by using the proposed XML DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE XIC SYSTEM "xic.dtd">
<XIC>
  <IC LBL="line.L1">
    <F NAME="pnt"> <OREF REFID="P1" /> </F>
    <F NAME="dir"> <OREF REFID="V1" /> </F>
  </IC>
  <IC LBL="cartesian_point.P1">
    <F NAME="coordinates">
      <AGG> <REAL VAL="0.0" /> <REAL VAL="0.0" /> <REAL VAL="0.0" />
      </AGG> </F>
  </IC>
  <IC LBL="vector.V1">
    <F NAME="orientation"> <OREF REFID="D1" /> </F>
    <F NAME="magnitude"> <REAL VAL="2.5" /> </F>
  </IC>
  <IC LBL="direction.D1">
    <F NAME="direction_ratios">
      <AGG> <REAL VAL="1.0" /> <REAL VAL="1.0" /> <REAL VAL="0.0" />
      </AGG> </F>
  </IC>
</XIC>
```

Example 2: *(see section 4.2.3, page 78)*

The Information Container for an *IfcBeam* object shown in the original example, i.e.:

```
IfcBeam.12345 (
   GlobalID:"12345"
   OwnerHistory:oRef(IfcOwnerHistory.1175223)
   LocalPlacement:oRef(IfcLocalPlacement.9731211)
   calcBeamSectionArea:1200.0
   Label:"BEAM-1" )
```

can be represented in the following way by using the proposed XML DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE XIC SYSTEM "xic.dtd">
<XIC>
   <IC LBL="IfcBeam.12345">
      <F NAME="GlobalID"> <STR VAL="12345" /> </F>
      <F NAME="OwnerHistory">
         <OREF REFID="IfcOwnerHistory.1175223" /> </F>
      <F NAME="LocalPlacement">
         <OREF REFID="IfcLocalPlacement.9731211" /> </F>
      <F NAME="calcBeamSectionArea"> <REAL VAL="1200.0" /> </F>
      <F NAME="Label"> <STR VAL="BEAM-1" > </F>
   </IC>
<XIC>
```

Example 3: *(see section 4.3.4, page 84)*

In order to use XML for recording a client/server session with PROMISE, it is necessary to modify the definition of the **XIC** element and to introduce two new elements as follows:

```
<!-- Modified definition for XIC:
     changed model, added attribute declarations -->
<!ELEMENT XIC (IC | REQ | RESPONSE)+ >
<!ATTLIST XIC
          LBL CDATA #IMPLIED
          CONTENT-TYPE (SESSION | REQUEST | RESPONSE | IC) "IC" >
<!-- Additional elements for the C/S data exchange
     capturing client requests (tag: REQ)
     and server responses (tag: RESPONSE).
     This allows to record automatically a session. -->
<!ELEMENT REQ (%icValue;)? >
<!ATTLIST REQ
          %label;
          ID      ID      #REQUIRED
          CONCEPT NMTOKEN #REQUIRED
          OID     CDATA   #REQUIRED
          LOCALID CDATA   #IMPLIED
          METH    NMTOKEN #REQUIRED
          MODE    (sync | async) "sync" >
```

```
<!ELEMENT RESPONSE (%icValue;)? >
<!ATTLIST RESPONSE
          %label;
          REQREF  IDREF   #REQUIRED
          CONCEPT NMTOKEN #IMPLIED
          OID     CDATA   #IMPLIED
          LOCALID CDATA   #IMPLIED
          METH    NMTOKEN #IMPLIED
          STATUS  NMTOKEN #IMPLIED
          EXCEPTION-MESSAGE CDATA #IMPLIED >
```

Using this extended definition, the request/response sequence of the original example, i.e.:

```
Input> Request1(session:oRef(session.1)
                OID:"MODEL.STRUCT_1:IfcBeam.232"
                meth:"getAttribute"
                inParams:InfoContainer(attName:"GenericType")
                localID:"BEAM-1"
                syncMode:sync
                status:sent)
Sync request accepted. Request name: "Request.43" .
Server response:
Response1(
  responseTo:oRef(Request.43)
  status:ok
  content:outParams(attName:"GenericType" attContent:TRUSS)
  session:oRef(session.1)
  OID:oRef(MODEL.STRUCT_1:IfcBeam.232)
  localID:"BEAM-1"
  meth:"getAttribute")
```

can be recorded as a XML document e.g. as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE XIC SYSTEM "xic.dtd">
<XIC LBL="session.1" CONTENT-TYPE="SESSION">
  <REQ LBL="Request1" ID="43" CONCEPT="IfcBeam"
       OID="MODEL.STRUCT_1:IfcBeam.232" LOCALID="BEAM-1"
       METH="getAttribute" MODE="sync" >
    <IC LBL="InfoContainer">
        <F NAME="attName"> <STR VAL="GenericType" /> </F>
    </IC>
  </REQ>
  <RESPONSE LBL="Response1" REQREF="43" STATUS="OK">
    <IC LBL="outParams">
        <F NAME="attName" <STR VAL="GenericType" /> </F>
        <F NAME="attContent"> <ENUM VAL="TRUSS" /> </F>
    </IC>
  </RESPONSE>
</XIC>
```

Note that this suggested extension of the DTD would not influence in any way an implementation of the Information Container model using the original specification shown in section III.1 above.

Appendix IV XML DTD for CSML Externalisation

Similar to Information Containers, a mapping specification in CSML can easily be converted to XML due to the structured, declarative character of the CSML language. The advantages of this alternative syntax for mapping specifications are, as by Information Containers, in the easy parsing of XML data, the non restrictive representation format and the possibility to view a mapping with a standard WWW-Browser, making full use of available hypertext techniques. Its disadvantages are also similar to the XML-based externalisation of Information Containers. However, there are also two important differences.

First, a mapping specification requires a rich set of data types including the symbolic representation of entity classes, attributes and functions which is not provided by XML. This can be partially overcome by using the *XML Schema* specification proposed by W3C (Fallside et al. 2000) but it would lead to an extremely verbose and convoluted representation with almost no added value to the design of correct mappings, neither to the mapping process itself. However, without such capabilities, the reverse conversion of a mapping from XML to CSML, as well as its correct, unambiguous parsing are not possible. This sets the scope of XML usage to an **output presentation format for CSML**.

Second, whilst Information Containers are primarily intended to support client-server communication, the role of CSML is on conceptual level, complementing the data model schemas used in the environment. There is no need to exchange mapping specifications in a running system since a mapping operation (represented in Information Container format) only *references* a CSML-based specification, but does not include it in its data arguments.

Therefore, the primary benefit of using XML as an alternative format for mappings is in **supporting the mapping development process** - for authoring, discussion, coordination and documentation purposes. In accordance with that, the rationale of the proposed XML DTD is (1) to enable the automatic generation of XML data from existing CSML specifications, and (2) to provide features for cross-linking the mapping specification with the underlying EXPRESS schemas, function definitions and other related mappings.

IV.1 Formal specification of the XML DTD for CSML Externalisation

The following DTD presents the proposed externalisation of CSML in XML format. Unlike the DTD for Information Container externalisation it does not define specific constructs for each of the data types that may appear in a mapping specification. Its focus is on the automatic generation of a XML document from a *correct* CSML specification, and not on the validation of that specification itself. Thus, it provides a compact, straightforward representation of the main constructs of CSML, whereas the rich set of available lexical elements in CSML is basically mapped to human readable character data without specific care of their formal semantics. Consequently, all basic data types are mapped to **`#PCDATA`** in the content models of the respective XML elements, and as **`CDATA`** or **`NMTOKEN`**s in attribute definitions, whereas high-level constructs are kept as similar as possible to the corresponding CSML definitions. However, due to the specific syntactic features of XML, several deviations from the CSML constructs were nevertheless necessary, as described below:

- To allow for the representation of variables as **NMTOKEN**s and at the same time distinguish them from schema elements like class/attribute names etc., the definition of variables is changed
  from: **variable = '?' symbol**
  to: **variable = '_' symbol**
  and the syntax for keywords is changed respectively
  from: **keyword = ['_'] symbol**
  to: **keyword = ['_ _'] symbol**.
- For similar reasons, the addressing operator '**->**' is substituted by '**.**'. Thus, in XML a complex attribute reference of the form **schema:class->attribute** will be represented by **schema:class.attribute**, and **class->?variable** will be represented by **class._variable** respectively.
- Simple templates and literals are all represented as **<SRC>** elements, and all kinds of expressions are represented as **<EXPR>** elements, both with **#PCDATA** content.
- Positional CSML constructs, as well as keywords that may appear in different context, are defined by *globally unique tags* in the XML DTD as required by the XML syntax. This applies to the tags **<MAP-SCHEMA>**, **<MAP-CLASS>**, **<COPY-CLASS>**, **<MAKE-VAR>**, **<REL>**, **<GROUPING>**, **<EXCLUDE>** and **<FN>**.
- The target of a class or attribute mapping declaration (which is also positional in CSML) is provided by means of a **TARGET** attribute of type **NMTOKENS**.
- The representation of comments is dropped since XML supports the definition of comments at any place.
- Finally, a few keywords of CSML are renamed for better readability in the context of a XML document.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Base element: enables the representation
     of multiple schema mapping specifications
     in a single XML document. -->
<!ELEMENT CSML (MAP-SCHEMA)+ >
<!ATTLIST CSML TITLE CDATA #IMPLIED >
<!-- General definition for the representation of terms.
     Note:
     Template and term operators are adopted from CSML as is,
     whereas variable, simpleTemplate, literal and NUL are all
     mapped to <SRC>, and  expressions are mapped to <EXPR>.
     The practical implementation of the latter may use PIs
     in place of CDATA. -->
<!ENTITY % term "SRC | EXPR |
                 APPLY | ASSOC | DESCENDANTS |
                 ITERATE-ON | MAPCAR | NEW | REF |
                 USER-CHOICE | USER-INPUT |
                 ONEOF | LISTOF | SETOF | MAX | MIN" >
<!-- General definition for the representation of functions. -->
<!ENTITY % fn-decl "(FN | EXPR), ARGS?" >
```

```
<!-- Parameter entities for attributes
     representing common CSML constructs -->
<!ENTITY % classes   "CLASS   NMTOKENS  #REQUIRED" >
<!ENTITY % froms     "FROM    NMTOKENS  #REQUIRED" >
<!ENTITY % targets   "TARGET  NMTOKENS  #REQUIRED" >
<!ENTITY % assigns   "ASSIGN  NMTOKENS  #REQUIRED" >
<!ENTITY % for-attrs "FOR     NMTOKENS  #IMPLIED"  >
<!ENTITY % ref       "REF     NMTOKEN   #IMPLIED"  >
<!ENTITY % valueType "TYPE    NMTOKEN   #REQUIRED" >
<!ENTITY % condType  "TYPE    (ALL | ONEOF) 'ALL'" >
<!-- Common constructs for hyperlinks -->
<!ENTITY % id        "ID        ID      #IMPLIED"  >
<!ENTITY % attref    "ATTREF    IDREF   #IMPLIED"  >
<!ENTITY % fnref     "FNREF     IDREF   #IMPLIED"  >
<!ENTITY % classrefs "CLASSREFS IDREFS  #IMPLIED"  >
<!-- General definition for lists of hypertext links that
     can be used with lists of classes, attributes etc.
     e.g. in the form "see: http://... http://...".
     Uses a simplified version of the HTML element <A>. -->
<!ELEMENT LINKS (#PCDATA | A+)* >
<!ELEMENT A (%text;)+ >
<!ATTLIST A
          %id;
          HREF %URI; #IMPLIED >
<!ENTITY % text "#PCDATA" >
<!ENTITY % URI  "CDATA" >
<!--
  Schema mapping header
-->
<!ELEMENT MAP-SCHEMA (((CLASSES+) | DEPENDENT-CLASSES |
                      DEPENDENTS | PRESETS), (LINKS?)) >
<!ATTLIST MAP-SCHEMA
          LBL CDATA #IMPLIED
          %id;
          %froms;
          %targets;
          PARTIALLY (YES | NO) "NO"
>
<!--
  Top-level constructs
-->
<!ELEMENT CLASSES (COPY-CLASS | MAP-CLASS)+ >
<!ELEMENT DEPENDENT-CLASSES (COPY-CLASS | MAP-CLASS)+ >
<!ELEMENT DEPENDENTS (COPY-CLASS | MAP-CLASS)+ >
<!ELEMENT PRESETS (PLUG-IN-FN | MAKE-VAR)+ >
<!ELEMENT PLUG-IN-FN (#PCDATA | A)* >
<!ATTLIST PLUG-IN-FN
          %id;
          RUN  (BEFORE | AFTER | WHEN-CALLED) "WHEN-CALLED"
          SRC  %URI; #IMPLIED
>
```

```
<!--
  Class level constructs
-->
<!ELEMENT COPY-CLASS (CONDITIONS | EXCLUSIONS)* >
<!ATTLIST COPY-CLASS
          %id;
          %froms;
          %targets; >
<!ELEMENT MAP-CLASS (GROUPS | VAR | CONDITIONS | EXCLUSIONS |
                     ATTRIBUTES+)+, (LINKS?)) >
<!ATTLIST MAP-CLASS
          %id;
          %froms;
          %targets; >
<!ELEMENT GROUPS (GROUPING)+ >
<!ELEMENT GROUPING (COND?, LINKS?) >
<!ATTLIST GROUPING
          %id;
          %assigns;
          %classes; >
<!ELEMENT VAR (MAKE-VAR)+ >
<!ELEMENT MAKE-VAR (FROM+ | CONSTRUCTOR) >
<!ATTLIST MAKE-VAR
          %id;
          %assigns; >
<!ELEMENT CONDITIONS (REL | PRED)+ >
<!ATTLIST CONDITIONS %condType; >
<!ELEMENT COND (REL | PRED) >
<!ATTLIST COND %condType; >
<!ELEMENT REL (#PCDATA) >
<!ELEMENT PRED (%fn-decl;) >
<!ELEMENT EXCLUSIONS (EXCLUDE)+ >
<!ELEMENT EXCLUDE (LINKS?) >
<!ATTLIST EXCLUDE
          %classes;
          %for-attrs;
          %classrefs; >
<!--
  Attribute mapping constructs
-->
<!ELEMENT ATTRIBUTES (SAME | MAKE)+ >
<!ELEMENT SAME (LINKS?) >
<!ATTLIST SAME
          %id;
          %froms;
          %targets; >
<!ELEMENT MAKE ( (FROM+ | CONSTRUCTOR), DEFAULT?, LINKS?) >
<!ATTLIST MAKE
          %id;
          %targets; >
```

```
<-- Basic representation of the source attributes
    in attribute mappings -->
<!ELEMENT FROM ( (%term;)+, (%suffix-op;)? ) >
<!ELEMENT CONSTRUCTOR (%fn-decl;) >
<!--
  Simple templates
-->
<!ELEMENT SRC ( #PCDATA | A )* >
<!ELEMENT EXPR ( #PCDATA ) >
<!--
  Term operators
-->
<!ELEMENT ONEOF (%term;)+ >
<!ELEMENT LISTOF (%term;)+ >
<!ELEMENT SETOF (%term;)+ >
<!ELEMENT MAX (%term;)+ >
<!ELEMENT MIN (%term;)+ >
<!--
  Template operators
-->
<!ELEMENT APPLY (%fn-decl;) >
<!ELEMENT ASSOC (COND?, LINKS?) >
<!ATTLIST ASSOC
          %classes;
          %for-attrs;
          %classrefs;
          %ref;
          %attref; >
<!ELEMENT DESCENDANTS (COND?, LINKS?) >
<!ATTLIST DESCENDANTS
          %classes;
          %for-attrs;
          %classref; >
<!ELEMENT ITERATE-ON ((%term;)+, COND?, LINKS?) >
<!ELEMENT MAPCAR (%fn-decl;) >
<!ELEMENT NEW (COND?, LINKS?) >
<!ATTLIST NEW
          %classes;
          %classrefs;
          %for-attrs;
          %ref;
          %attref; >
<!ELEMENT REF ((%term;)+, LINKS?) >
<!ATTLIST REF
          ATTR NMTOKEN #REQUIRED
          %attref; >
```

```
<!ELEMENT USER-CHOICE ((ONEOF | LISTOF), INIT+, DEFAULT?, LINKS?) >
<!ATTLIST USER-CHOICE
          PROMPT CDATA #IMPLIED
          RUN CDATA #IMPLIED >
<!ELEMENT USER-INPUT (INIT+, DEFAULT?, LINKS?) >
<!ATTLIST USER-INPUT
          PROMPT CDATA #IMPLIED
          RUN CDATA #IMPLIED
          %valueType; >
<!--
  Embedded constructs
-->
<!ELEMENT INIT ((%term;)+, LINKS?) >
<!ELEMENT DEFAULT ((%term;)+, LINKS?) >
<!ENTITY  % suffix-op "APPLY | MAPCAR | MAP" >
<!ELEMENT MAP (%fn-decl;) >
<!ATTLIST MAP %valueType; >
<!ELEMENT FN EMPTY >
<!ATTLIST FN
          NAME NMTOKEN #REQUIRED
          %fnref; >
<!ELEMENT ARGS ((%term;)+, LINKS?) >
<!-- End of the XML DTD for CSML externalisation -->
```

IV.2 Examples

To provide a better basis for comparison, five selected examples from chapter 6 are re-used. The first four examples apply to the representation of the relational operations shown in section 6.5, and the last is reproduced from the sixth mapping example shown in section 6.6.

Example 1: Projection *(see section 6.5.1, page 183)*

The principal CSML specification for a *projection*, i.e.:

```
(MAP CLASS T FROM S
   ATTRIBUTES (SAME t1 ... tm AS st1 ... stm )
 )
```

will be externalised in the following way on the basis of the proposed XML DTD:

```
<MAP-CLASS TARGET="T" FROM="S">
  <ATTRIBUTES>
    <SAME TARGET="t1 ... tm" FROM="st1 ... stm"/>
  </ATTRIBUTES>
</MAP-CLASS>
```

This example shows the close correspondence of the XML DTD to the native CSML-based structure of a mapping specification.

Example 2: Intersection *(see section 6.5.2, page 185)*

The principal CSML specification for an *intersection*, i.e.:

```
(MAP CLASS T FROM R S
   CONDITIONS
     ( $r_1$ = $s_1$ ) ... ( $r_k$ = $s_k$ )
   ATTRIBUTES
     (SAME $t_1$ ... $t_n$ AS $r_1$ ... $r_n$ )
 )
```

will be externalised in the following way on the basis of the proposed XML DTD:

```
<MAP-CLASS TARGET="T" FROM="R S">
  <CONDITIONS>
    <REL> $r_1$ = $s_1$ </REL>
    <REL> $r_k$ = $s_k$ </REL>
  </CONDITIONS>
  <ATTRIBUTES>
    <SAME TARGET="$t_1$ ... $t_n$" FROM="$r_1$ ... $r_n$"/>
  </ATTRIBUTES>
</MAP-CLASS>
```

The great similarity between the CSML and the XML representations is evident here as well. However, relations of the form $r_k = s_k$ are simply *ordinary text* in XML and cannot be directly interpreted by a parser.

Example 3: Telescope *(see section 6.5.4, page 189)*

The principal CSML specification for the virtual integration operation "*telescope*", i.e.:

```
(MAP CLASS T FROM R
   ATTRIBUTES
     (SAME $t_1$ ... $t_n$ AS $r_1$ ... $r_n$ )
     (MAKE $t_{n+1}$ FROM (REF $r_S$ FOR $s_1$))
     ...
     (MAKE $t_{n+p}$ FROM (REF $r_S$ FOR $s_p$ )))
```

will be externalised in the following way on the basis of the proposed XML DTD:

```
<MAP-CLASS TARGET="T" FROM="R">
  <ATTRIBUTES>
    <SAME TARGET="$t_1$...$t_n$" FROM="$r_1$...$r_n$"/>
    <MAKE TARGET="$t_{n1}$">
      <FROM> <REF ATTR="$r_S$"> <SRC> $s_1$ </SRC> </REF> </FROM>
    </MAKE>
    ...
    <MAKE TARGET="$t_{n+p}$">
      <FROM> <REF ATTR="$r_S$"> <SRC> $s_p$ </SRC> </REF> </FROM>
    </MAKE>
  </ATTRIBUTES>
</MAP-CLASS>
```

In this only a bit more complex example, the verbosity of the XML format and the lack of adequate capabilities for the representation of different data types can already be noticed (s_1 and s_p are just character data).

Example 4: Addition *(see section 6.5.4, page 189)*

The principal CSML specification for adding a new entity class to the target model in a mapping, i.e.:

```
(MAP CLASS T FROM S
   VAR (MAKE ?A CONSTRUCTOR F ARGS X Y ... )
   ATTRIBUTES
     (SAME ALL)
     (MAKE t_a FROM ?A ))
```

will be externalised in the following way on the basis of the proposed XML DTD:

```
<MAP-CLASS TARGET="T" FROM="S">
  <VAR>
    <MAKE-VAR ASSIGN="_A">
      <CONSTRUCTOR>
        <FN NAME="F"/>
          <ARGS> <SRC> x </SRC> <SRC> y </SRC> ...
          </ARGS>
      </CONSTRUCTOR>
    </MAKE-VAR>
  </VAR>
  <ATTRIBUTES>
    <SAME TARGET="ALL" FROM="ALL"/>
    <MAKE TARGET="t_a">
      <FROM> <SRC> _A </SRC>
      </FROM>
    </MAKE>
  </ATTRIBUTES>
</MAP-CLASS>
```

The differences between the CSML and the XML representation evident from the previous example are even more stronger here. This example also shows the "weak" representation of a function in the **`<CONSTRUCTOR>`** element, and the modified syntax for variables (**`_A`** instead of **`?A`**).

Example 5: adapted from (Amor 1997) *(see ex. 6, section 6.6, page 197)*

Along with most of the aforementioned issues, this somewhat longer example demonstrates the main advantages of the XML format, i.e. (1) the specification of hyper-links to other related resources, and (2) the capability to represent multiple mapping specifications in a single XML document.

The use of hyperlinks is illustrated only in the forward mapping specification, for the attributes **`height`** and **`width`** of the source entity **`wall`**, and for the two EXPRESS schemas themselves, but it can be easily extended for all other referenced elements.
The representation of both mappings in one XML document has no influence on the actual

mapping operations executed in a running environment but it provides advantages for publishing and discussing a mapping specification *during* its development.

CSML specification from the original example:

Mapping specification **shared_model -> local_model:**

```
(MAP local_model FROM shared_model
 CLASSES
  (MAP wall FROM wall
    ATTRIBUTES
      (MAKE area FROM
                 (APPLY (LAMBDA (X Y) (* X Y))
                        ARGS height width))
    )
 )
```

Inverse mapping:

```
(MAP shared_model FROM local_model
 CLASSES
  (MAP wall FROM wall
    ATTRIBUTES
      (MAKE height FROM (USER-INPUT
                         REAL "Height of wall ~: " OBJECTNAME))
      (MAKE width FROM (APPLY (LAMBDA (A H) (/ A H))
                              ARGS area (NEW wall FOR height)))
    )
 )
```

Corresponding XML representation:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE CSML SYSTEM "csml.dtd">
<CSML>
<MAP-SCHEMA LBL="Ex.5: Forward mapping shared_model->local_model"
            TARGET="local_model" FROM="shared_model">
  <CLASSES>
  <MAP-CLASS TARGET="wall" FROM="wall">
    <ATTRIBUTES>
      <MAKE TARGET="area">
        <FROM>
        <APPLY>
          <EXPR> (LAMBDA(X Y) (* X Y)) </EXPR>
          <ARGS>  <-- as hyperlinks to the resp. EXPRESS spec’s -->
          <SRC> <A HREF="shared_model.html#wall">height</A> </SRC>
          <SRC> <A HREF="shared_model.html#wall">width</A> </SRC>
          </ARGS>
        </APPLY>
        </FROM>
      </MAKE>
    </ATTRIBUTES>
  </MAP-CLASS>
  </CLASSES>
  <LINKS>
  <-- hypertext references to the resp. EXPRESS schema spec’s -->
  See: <A HREF="shared_model.html">shared_model</A> and
       <A HREF="local_model.html">local_model</A>.
  </LINKS>
</MAP-SCHEMA>
<MAP-SCHEMA LBL="Ex.5, Inverse mapping local_model->shared_model"
            TARGET="shared_model"
            FROM="local_model">
  <CLASSES>
  <MAP-CLASS TARGET="wall" FROM="wall">
    <ATTRIBUTES>
      <MAKE TARGET="height">
        <FROM>
        <USER-INPUT PROMPT="Height of wall ~ " TYPE="REAL">
             <INIT> <SRC> <?csml OBJECTNAME ?> </SRC> </INIT>
        </USER-INPUT>
        </FROM>
      </MAKE>
      <MAKE TARGET="width">
        <FROM>
        <APPLY>
          <EXPR> (LAMBDA(A H) (/ A H)) </EXPR>
          <ARGS> <SRC> area </SRC>
                 <NEW CLASS="wall" FOR="height"/> </ARGS>
        </APPLY>
        </FROM>
      </MAKE>
    </ATTRIBUTES>
  </MAP-CLASS>
  </CLASSES>
</MAP-SCHEMA>
</CSML>
```

Appendix V Prototyped Project Data Management Operations

This appendix provides an overview of the currently prototyped basic operations supported by the developed project data server, PROMISE.

The content is structured in four equally formatted tables (V.1 to V.4) as follows:

- in the first column, the names of the respective operations are given, grouped by object classes and sorted alphabetically within each class; operations marked with an asterisk (`*`) are valid only for the IFC Project Model, all other operations are applicable for any EXPRESS-based models that might be used in the environment.
- the second column, C/I, indicates if the operation is applicable to a class (C), its instances (I), or both;
- the third column provides a short description of the operation including its purpose, the input and output arguments, and the user access rights required for its execution.

In addition, a summary table of all operations (table V.5) details the type of each operation, the triggered model state transitions and the permissible execution modes (synchronous/asynchronous, abbreviated to *sync/async* respectively), as introduced in principle in section 4.6.

Notes:

1. The **data types of the arguments** of each operation conform to the *Information Container* specification presented in section 4.2. Optional arguments, indicated by the keyword OPT have the same meaning as in EXPRESS-C.
2. When using an operation in **TCP/IP-based requests**, the name of the addressed class or instance and the name of the operation should be provided in the *OID* parameter and the *meth* parameter of the *Request* object respectively, and the input parameters of the operation should be provided as the value of the parameter *inParams*, packaged in an *InfoContainer* as name-value tupples. The output is returned in a *Response* object, packaged in an *InfoContainer* as the value of the parameter *content*. An example of this kind of usage of the prototyped operations is given in section 4.3.
3. When using an operation with the **Java API** presented in Appendix II, the generic *execMeth* method should be called for the required class or instance, with the name of the operation as the first parameter, and the values of the input parameters packaged in an *InfoContainer*, as the second parameter of the method call, respectively. The output is returned also as an *InfoContainer*, and can then be individually processed with the help of the methods provided by the Java API. An example of this usage is given in section II.3.
4. If used in a **CORBA-based environment**, an operation should be declared as an IDL **`interface`**, with **`in`** and **`out`** arguments corresponding to the *in* and *out* arguments from the tables below (the IDL **`inout`** argument type is not used). The mapping of the elements of the *InfoContainer* schema to respective CORBA data types is relatively easy to achieve; some more difficulty provides only the *symbol* type which may e.g. be mapped to a **`scoped_name`** CORBA type (cf. OMG 1998). In this way, an implementation of any operation in another programming language, such as C/C++, Smalltalk or Ada, can be realised (see e.g. Orfali et al. 1997).
5. The mapping to EXPRESS-C is done by using the *InfoContainer* schema. The *symbol* data type (not found in EXPRESS) is mapped to a *defined type* with underlying `STRING` type to distinguish it from the “normal” base string type.

6. By any kind of usage of an operation, the **overall result** and/or eventual error messages are always returned in the high-level arguments of the *Response*, whereas the **output parameters** are all packaged in the attribute *content.*
7. The specified **access rights** for each operation indicate the *user roles* which are privileged to execute the operation; *f(ACL)* means that the access rights of the user or application are determined on the basis of the *Access Control List* for the current object; *owner* means that the operation can be executed only by the primary model user, e.g. an architect for an architectural domain model; *admin* means that the operation is allowed only for users with administrator rights – normally, one person authorised by the project manager. The user roles are inheritable, i.e. *f(ACL)* automatically includes the owner, and *owner* automatically includes the administrator.
8. By many operations there is an output argument named *modelRef*. In such cases the operation returns one or more lists of objects which all belong to the same model. To reduce the output, the model reference is returned only in the argument *modelRef*, whereas all other *Obj_Ref*'s are provided in short form, i.e. without a **`modelID`** prefix. Similarly, several operations have an argument named *localFileRef*. When used as input, it indicates the location of a local file to be *uploaded* to the project data repository, and when used as output – the location to which a *downloaded* file should be saved. In both cases, the read/write permission should be set as appropriate at the client platform to enable the execution of the upload or download process respectively.

V.1 Generic top-level server operations for all modelling objects

The following table presents the operations implemented for the top-level meta model object **`Concept`**, and the top-level IFC class **`IfcRoot`** respectively.

All operations for **`IfcRoot`** are generic and are also valid for the meta model class `Concept`. Thus they are applicable for any defined entity in any data model, and not only for the entities of the IFC Project Model. **`IfcRoot`** merely emphasises that the operations shown in the table are valid for *all* IFC object classes.

Table V.1: Generic top-level server operations for all modelling objects

Operation	**C/I**	**Description**
Class: *Concept / IfcRoot*		
create	C	Creates an instance of the referenced class in the currently opened model from the server's project data repository, assigns it a globally unique ID, and returns the reference ID of the new instance. Arguments: in: content: *InfoContainer* out: objRef: *Obj_Ref* Access rights: *owner*
checkIn	I	Stores an object instance that has been checked out back into the respective model in the project data repository. If the model is in use by others, a new model version is created. Arguments: in: content: *InfoContainer* OPT status: *string* ("changed", "unchanged", "unknown") out: objRef: *Obj_Ref* Access rights: *owner*

Table V.1 (cont.): Generic top-level server operations for all modelling objects

Operation	C/I	Description
checkOut	I	Retrieves an object instance from the project data repository for local processing and sets the model in *LongWrite* state. This is the basic method to retrieve and modify arbitrary subsets of the model data from the project data server. However, this feature is enabled only for the model owner. Arguments: in: N/A out: attributes: *InfoContainer* (each attribute: name-value tupple) Access rights: *owner*
find	C	Searches for objects in the given class satisfying the search criteria and returns their IDs for use in subsequent requests. A search criterion may be e.g. an attribute value, a range of values, a predicate over the attribute values combined with a boolean operation etc. (see section 4.7) Arguments: in: searchExpr: *lisp_expr* (as string) OPT forModel: *Obj_Ref* out: resultSet: *aggregation(valueSelect)* (the type of 'valueSelect' depends on the query) Access rights: *f(ACL)*
getAccessRights	I	Returns the access rights of the user specified in the argument *forUser* w.r.t. the referenced object. However, currently the access rights are set on model level. This operation is provided basically for future extensions. Arguments: in: OPT forRole: *symbol* OPT forUser: *Obj_Ref* out: accessRights: *aggregation(symbol)* Access rights: *f(ACL)*
getAttribute	I	Returns the value of the specified attribute of the referenced object. Arguments: in: attName: *symbol* out: attName: *symbol* attContent: *typedValueSelect* Access rights: *f(ACL)*
getInstances	C	Returns all instances of the referenced class in the current model, or in another model, respectively specified in the parameter *forModel*. Arguments: in: OPT forModel: *Obj_Ref* out: modelRef: *Obj_Ref* objRefs: *aggregation(Obj_Ref)* Access rights: *f(ACL)*

Table V.1 (cont.): Generic top-level server operations for all modelling objects

Operation	C/I	Description
getMethod	C/I	Returns the argument names and types of the specified operation. This is basically a *reflection* possibility that may be useful for advanced applications. Arguments: in: methName: *symbol* out: methParams: *InfoContainer* (each element: name-type tupple, as symbols) Access rights: *f(ACL)*
getMethods	C/I	Returns the names of all operations on the referenced object that are accessible to the logged in user. Arguments: in: N/A out: methNames: *aggregation(symbol)* Access rights: *f(ACL)*
getRelationships	I	Returns the relationships of the referenced object to/from other objects in the current model. By default, all “direct” pointers to/from other objects are examined. The search can be constrained – by specifying the set of classes to be considered, or expanded – by specifying a deeper level for the relationship graph (the default depth is 1, a depth of –1 means “the whole relationship graph”). Arguments: in: OPT classRefs: *aggregation(Obj_Ref)* OPT depth: *longint* (default: 1) out: relating: *aggregation(Obj_Ref)* related: *aggregation (aggregation(Obj_Ref))* (nested level depth depends on context) Access rights: *f(ACL)*
inspect	C/I	Returns the content of the referenced object as an InfoContainer. This is the basic function to examine any kind of modelling objects on the project data server. However, in order to be processed, an object has to be first *checked out*. ‘Inspect’ provides only a viewing capability. When *inlineRef* is true, *attributes* shall contain the full representation of the object instances referenced by the “inspected” object, and not only their *Obj_Ref*s. However, such output can be quite verbose. Arguments: in: OPT inlineRef: *boolean* out: *for Classes:* parentRef: *Obj_Ref* childRefs: *aggregation(Obj_Ref)* attributes: *aggregation(symbol)* *for Instances:* parentRef: *Obj_Ref* attributes: *InfoContainer* (each attribute is a feature, i.e. a name-value tupple) Access rights: *f(ACL)*

Table V.1 (cont.): Generic top-level server operations for all modelling objects

Operation	C/I	Description
retrieve	I	Similar to 'inspect', but returns the object's data in more concise form. This operation is limited only to *viewing* the content of an object. Arguments: in: N/A out: attributes: *InfoContainer* Access rights: *f(ACL)*
setAccessRights	C/I	Sets the access rights for the specified user (see *getAccessRights*). Arguments: in: forRole: *symbol* forUser : *Obj_Ref* accessRights: *aggregation(symbol)* out: N/A Access rights: *owner*
setAttribute	I	Sets the value of the specified attribute of the referenced object to the value given in *attContent*. Arguments: in: attName: *symbol* attContent: *valueSelect* out: N/A Access rights: *owner*
status	I	Returns the status and the version history of the referenced object. The status information is presented according to the *IfcUtilityResource* schema (see IAI 1999c). Arguments: in: OPT forAllVersions: *boolean* out: currentStatus: *aggregation(string)* OPT versions: *aggregation(Obj_Ref)* OPT versionStatus: *aggregation (aggregation(string))* Access rights: *f(ACL)*
testAttribute	I	Tests if the specified attribute of the referenced object is set. Arguments: in: attName: *symbol* out: isSet: *logical* Access rights: *f(ACL)*
unsetAttribute	I	Unsets (i.e. sets to 'Unknown') the value of the specified attribute. Arguments: in: attName: *symbol* out: N/A Access rights: *owner*
update	I	Updates the content of an entity instance. The arguments and access rights are the same as by the *create* operation.

V.2 Specific server operations for *IfcKernel* objects

Table V.2 presents operations that have been specifically designed for IFC-based implementation. All these operations are defined at the Kernel Model level, and are therefore available also for any domain/application model extensions.

At the time when they were developed, the available IFC version was 1.5 final. Efforts have been made to adapt these operations to the current IFC version; however, as this has been done late, and with not much time available, the author apologises for possibly existing inconsistencies that might have remained undiscovered.

Table V.2: Specific server operations for *IfcKernel* objects

Operation	**C/I**	**Description**
Class: *IfcRelationship*		
getRelatingObject*	I	Returns the "Relating" object referenced through the specified "relationship" object. This operation is actually a shortcut for *getAttribute*, with parameter *attName* = "Relating". Arguments: in: N/A out: modelRef: *Obj_Ref* objRef: *Obj_Ref* Access rights: *f(ACL)*
getRelatedObjects*	I	Returns the "Related" objects referenced through the specified "relationship" object. Arguments: in: N/A out: modelRef: *Obj_Ref* objRefs: *aggregation(Obj_Ref)* Access rights: *f(ACL)*
getRelations*	I	Returns the "Relating" and "Related" objects referenced through the specified "relationship" object. Arguments: in: N/A out: modelRef: *Obj_Ref* relating: *Obj_Ref* related: *aggregation (Obj_Ref)* Access rights: *f(ACL)*
Class: *IfcRelGroups*		
addToGroup*	I	Adds an object to the referenced "group" object. Arguments: in: objRefs: *aggregation(Obj_Ref)* out: N/A Access rights: *owner*
removeFromGroup*	I	Removes an object from the referenced "group" object. Arguments: in: objRefs: *aggregation(Obj_Ref)* out: N/A Access rights: *owner*

Table V.2 (cont.): Specific server operations for *IfcKernel* objects

Operation	C/I	Description
Class: *IfcObject*		
getDocAssignment*	I	Returns a list of references to document URLs associated with this model (see IAI 1999c). Arguments: in: N/A out: modelRef: *Obj_Ref* DocRefs: *aggregation(string)* Access rights: *f(ACL)*
getPropertySets*	I	Returns the "property sets" associated with the given object. The result set can optionally be restricted by specifying the type of property sets for which the search should be performed (this type is assumed to be found in the attribute *Name* of *IfcProperty* and its respective subclasses – *IfcEnumeratedProperty*, *IfcPropertyList*, *IfcSimpleProperty*, *IfcSimplePropertyWithUnit*, *IfcObjectReference*). Arguments: in: OPT classifier: *aggregation(string)* out: modelRef: *Obj_Ref* objRefs: *aggregation(Obj_Ref)* Access rights: *f(ACL)*
getRelations*	I	Finds all "Related" objects to the referenced object, specified with the help of "relationship" objects. The output argument *relating* contains instances of subtypes of *IfcRelationship* in which the given object plays the role "Relating", and the argument *related* contains a list of respective aggregations, comprising all the "Related" objects. Arguments: in: N/A out: modelRef: *Obj_Ref* relating: *aggregation(Obj_Ref)* related: *aggregation (aggregation(Obj_Ref))* Access rights: *f(ACL)*
Class: *IfcProject*		
merge*	I	Merges two or more model versions associated with this "project" object, and returns the reference to the new "merged" model. In fact, due to the specific architecture of the IFC modelling framework, an *IfcProject* instance is the basic reference to all models associated with a given project. Therefore, merge is actually a typical model-level operation,. accordingly defined for the *MODEL* entity class (see Table V.3). Its use with *IfcProject* is merely for convenience. Arguments: in: baseModel: *Obj_Ref* targetModelRefs: *aggregation(Obj_Ref)* OPT mergePolicy: *symbol* out: modelRef: *Obj_Ref* Access rights: *admin*

Table V.2 (cont.): Specific server operations for *IfcKernel* objects

Operation	**C/I**	**Description**
Class: *IfcProduct*		
view*	I	Returns a BLOB containing the geometric representation of the referenced product object in a format suitable for processing with a visualisation or CAD tool, and optionally downloads that BLOB to the location given in *LocalFileRef*. Currently, only 'Bounding Box' geometry is supported, i.e. *representationType* defaults to *BoundingBox*. As the output file is "partial", i.e. contains data only about the referenced entity, it is on the responsibility of the application to interpret it correctly in the context of its other data. The view types TEXT, HTML and XML are provided for future extensions. Arguments: in: viewType: *symbol* (one of DXF, VRML, TEXT, HTML) OPT representationType: *symbol* OPT localFileRef: *BLOB_Ref* out: repositoryRef: *BLOB_Ref* Access rights: *f(ACL)*

V.3 Generic model-level server operations

Table V.3 presents the operations applicable to a *whole* model. They enable global model management functions, such as *checkIn*, *checkOut*, *map*, *match* etc., in the same way as when individual entities are addressed. To invoke any of these operations, a respective instance of the Meta model object ***MODEL*** has to be referenced in the request.

Table V.3: Generic model-level server operations

Operation	**C/I**	**Description**
Class: *MODEL*		
abortTransaction	I	Aborts the current transaction for the referenced model. Arguments: N/A Access rights: *f(ACL)*
beginTransaction	I	Begins a transaction (request/response block) for the ref. model. Arguments: in: OPT mode: *symbol* (one of R, W, LR, LW, CI, MRG – see sect. 4.6) out: N/A Access rights: *f(ACL)*
checkIn	I	Uploads a model that has been checked out for local processing back to the project data repository. If specified, the argument *spf* should reference a local STEP physical file containing the model data. This argument controls the way in which the operation is performed: - if not present, *checkIn* assumes that the model has not been modified and "unlocks" it, without creating a new version (in this case, none of the other input arguments need to be provided); - if *spf* is specified, *checkIn* assumes that the model has been modified and uploads it to the project data repository, creating a new model version and optionally *matching* it with the latest existing version of the model.

Table V.3 (cont.): Generic model-level server operations

Operation	C/I	Description
checkIn (cont.)		The argument *verbose* controls the amount of the output data. If not provided or *false*, only the result status is returned. Otherwise, *checkIn* returns three parallel lists containing the object references of all objects in the checked in model, their respective numbers in the STEP physical file, and the status of each object (in the case that model matching has been performed as well). Arguments: in: OPT spf: *BLOB_Ref* OPT verbose: *boolean* OPT noMatch: *boolean* out: OPT objRefs: *aggregation(Obj_Ref)* OPT objNums: *aggregation(longint)* OPT objStatus: *aggregation(string)* (one of: “Same”, “New”, “Changed”, “NewOrChanged”,“Unknown”) Access rights: *f(ACL)*
checkOut	I	This is the basic method to retrieve and modify a whole model stored in the project data repository maintained by PROMISE. It checks out the model for local processing and sets a *LongWrite* lock. After a *checkOut*, only *checkIn* and *rollback* (in that case equal to “abort”) are available for the user who checked out the model. If the argument *retrieve* is set to *false*, the model is not downloaded, but only checked out and the *LongWrite* lock is set. This option is useful when the model is already available locally but has not yet been processed and is not yet locked. Arguments: in: OPT retrieve: *boolean* OPT localFileRef: *BLOB_Ref* out: repositoryRef: *BLOB_Ref* Access rights: *f(ACL)*
checkConsistency	I	Proves if all model data are consistent with respect to the underlying EXPRESS specification, i.e. if there are no dangling links, empty, but required attributes etc. Optionally, when *targetModelRef* is provided, the current model can be compared for consistence with another model used as “reference”. In that case, a *matching* is performed, and the models are assumed consistent if there are no differences found. This operation is not yet sufficiently tested. Arguments: in: OPT targetModelRef: *Obj_Ref* OPT verbose: *boolean* out: OPT objRefs: *aggregation(Obj_Ref)* OPT objStatus: *aggregation(string)* Access rights: *f(ACL)*
closeModel	I	Closes an opened model. Arguments: N/A Access rights: *f(ACL)*

Table V.3 (cont.): Generic model-level server operations

Operation	C/I	Description
commitPrepare	I	Sends a "prepare to commit" request to the project data server (this operation is only needed if a safe two-step commit protocol is implemented in the environment). Arguments: in: N/A out: N/A Access rights: *f(ACL)*
commit	I	Commits all changes made to the referenced model and ends the open transaction. A *commit* is final, and cannot be undone. However, without *commitPrepare*, it is not safe w.r.t. possible system crashes. Arguments: in: N/A out: N/A Access rights: *f(ACL)*
create	C	In principle, this operation is quite similar to *checkIn*. It is needed whenever a model is “checked in” for the first time. The argument *spf* is mandatory, as well as a user-defined model name. Also, unlike *checkIn*, *create* allows to specify a “reference” model (optionally based on another domain model schema). In that case, the newly created model is compared with this reference model, using the provided mapping specification, and all equivalent objects (in the sense of the mapping transformation) are assigned the same IDs as in the reference model, whereas the rest are given *new* IDs. However, this use of *create* is only possible if the mapping is not interactive, and may not succeed by complicated mappings. The output of *create* is identical to that of *checkIn*. Arguments: in: spf: *BLOB_Ref* modelName: *string* OPT refModel: *Obj_Ref* OPT mappingRef: *Obj_Ref* OPT verbose: *boolean* (see *checkIn*) out: modelRef: *Obj_Ref* OPT objRefs: *aggregation(Obj_Ref)* OPT objNums: *aggregation(longint)* OPT objStatus: *aggregation(string)* (see *checkIn*) Access rights: *owner*
delete	I	Deletes a model from the project data repository (can be executed only by a user with administrator privileges). Arguments: in: N/A out: N/A Access rights: *admin*

Table V.3 (cont.): Generic model-level server operations

Operation	C/I	Description
find	C/I	Searches for objects in the referenced model that satisfy the given search criteria. This operation is basically the same as the respective *find* operation defined for *Concept*. However, here the search criteria are applied to *all* object instances in the model, and not only to the instances of one object class. Arguments: in: searchExpr: *lisp_expr* (as string) out: resultSet: *aggregation(valueSelect)* (the type of 'valueSelect' depends on the query) Access rights: *f(ACL)*
getAccessRights	I	Returns the access rights of the user specified in the argument *forUser* w.r.t. the referenced model. Arguments: in: OPT forRole: *symbol* OPT forUser: *Obj_Ref* out: accessRights: *aggregation(symbol)* Access rights: *f(ACL)*
getAllModels	C	Returns a list of all available models in the project data repository which are currently associated with the referenced model schema. Arguments: in: N/A out: modelRef: *aggregation(Obj_Ref)* modelSchema: *aggregation(Obj_Ref)* Access rights: *f(ACL)*
getGroups*	I	Returns a list of references (Obj_Ref 's) for all instances of the subclasses of *IfcGroup* contained in the model (valid only for IFC models). Arguments: in: N/A out: modelRef: *Obj_Ref* objRefs: *aggregation(Obj_Ref)* (contains references to instances of any of the subclasses of IfcGroup) Access rights: *f(ACL)*
getModelVersions	C/I	Similar to *getAllModels*, but returns only the available versions of the referenced model or model schema.
getObjects	I	Returns a list of *Obj_Ref*s for all object instances contained in the model. Arguments: in: N/A out: modelRef: *Obj_Ref* objRefs: *aggregation(Obj_Ref)* Access rights: *f(ACL)*

Table V.3 (cont.): Generic model-level server operations

Operation	C/I	Description
getProductObjects*	I	Returns a list of Obj_Ref 's for all instances of the subclasses of *IfcProduct* contained in the model (valid only for IFC models). Arguments: in: N/A out: modelRef: *Obj_Ref* objRefs: *aggregation(Obj_Ref)* Access rights: *f(ACL)*
map	I	Performs a mapping for the referenced model using the mapping specification addressed through the argument *mappingRef*. *Map* can be used as an interactive operation. If not aborted, it will always create a new (target) model version. The argument *mode* controls the manner in which the mapping is performed, and the argument *verbose* controls the amount of output produced by the operation in a similar way as by *checkIn*. Arguments: in: mode: *symbol* (one of *interactive, partial, complete*) mappingRef: *Obj_Ref* OPT verbose: *boolean* OPT *localFileRef: BLOB_Ref* out: modelRef: *Obj_Ref* OPT objRefs: *aggregation(Obj_Ref)* OPT objStatus: *aggregation(string)* Access rights: *f(ACL)*
mapTry	I	This operation is essentially the same as *map*, except that it creates a temporary model. It can be useful to check if a *map* operation, that might require quite a long time to execute, would be successful. *mapTry* performs fewer checks and is faster than *map*. All arguments and access rights are as by *map*.
match	I	Performs a matching of two model versions based on the same model schema. The first model is the one on which the operation is invoked, and the second model is the one specified as *refModel*. The argument *verbose* controls the output produced by the operation: if *verbose* is *false* or not specified, only the detected new and changed objects are returned as lists of Obj_Ref 's in the output arguments *newObjects* and *changedObjects* respectively; if *verbose* is *true*, the status of all object instances is returned in a similar way as by *checkIn*. By default, a matching of all objects contained in the model is performed. However, it can be restricted only to the instances of the object classes specified in the optional argument *inclusionList.* Arguments: in: refModel: *Obj_Ref* OPT verbose: *boolean* OPT inclusionList: *aggregation(Obj_Ref)* out: OPT newObjects: *aggregation(Obj_Ref)* OPT changedObjects: *aggregation(Obj_Ref)* OPT objRefs: *aggregation(Obj_Ref)* OPT objStatus: *aggregation(string)* Access rights: *f(ACL)*

Table V.3 (cont.): Generic model-level server operations

Operation	C/I	Description
merge	I	This operation is conceptually identical to the *merge* operation specified for *IfcProject*, but is available for any data model, and not only for IFC-based models. It always locks the whole data store (see section 4.6). Arguments: in: targetModelRefs: *aggregation(Obj_Ref)* OPT mergePolicy: *symbol* out: modelRef: *Obj_Ref* Access rights: *admin*
openModel	I	Opens the referenced model for subsequent transactions. *openModel* must always be executed *before* performing any other operations on the respective model or its objects. Once opened, a model can be subject to an unlimited number of transactions before it is closed. Arguments: N/A Access rights: *f(ACL)*
retrieve	I	This operation is similar in function to *checkOut*, but the model is set in Read or LongRead state, depending on the execution mode (sync/async). *Retrieve* can be used only for viewing the content of the model locally; for modifying it has to be checked out. Arguments: in: OPT localFileRef: *BLOB_Ref* out: repositoryRef: *BLOB_Ref* Access rights: *f(ACL)*
rollback	I	Allows to undo all operations on the model, performed after it has been opened with *beginTransaction*. However, whilst conceptually an unlimited number of operations can be undone, in the prototype implementation this operation is restricted to the available amount of working memory. As a rule of thumb, the last executed operation can always be undone, the number of additional possible "undo" steps depends on the amount of data affected by the preceding operations. In any case, the previous consistent version of the model will at least be available for further work. Arguments: N/A Access rights: *f(ACL)*
setAccessRights	I	Sets the overall access rights for the specified user and/or role w.r.t. the referenced model. The specified access rights are automatically applied to *all* objects in the model. Arguments: in: OPT forRole: *symbol* OPT forUser: *Obj_Ref* accessRights: *aggregation(symbol)* out: N/A Access rights: *owner*

Table V.3 (cont.): Generic model-level server operations

Operation	**C/I**	**Description**
Class: *MODEL*		
view	I	Returns a BLOB which contains the geometric representations of the tangible modelling objects in the referenced model in a format suitable for processing with a visualisation or CAD tool, and optionally downloads it to the location specified in *LocalFileRef.* The concrete implementation of the *view* operation varies according to the underlying model schema. For IFC-based models, it is similar to the respective *view* operation defined for *IfcProduct* in Table V.2 above, but returns the respective geometric representations of *all* instances of the subclasses of *IfcProduct* contained in the model. Arguments: in: viewType: *symbol* (one of *DXF*, *VRML*, *TEXT*, *HTML*) OPT localFileRef: *BLOB_Ref* out: repositoryRef: *BLOB_Ref* Access rights: *f(ACL)*
Class: *ModelSchema*		
getMappings	I	Returns the references to *all* available mapping specifications defined for this model schema. Arguments: in: N/A out: mappingSchemas: *aggregation(Obj_Ref)* (contains references to instances of class MappingSchema) Access rights: *f(ACL)*
getModels	I	Returns the references to *all* available models in the project data repository that are associated with this model schema. Arguments: in: N/A out: modelRef: *aggregation(Obj_Ref)* (contains references to instances of class MODEL) Access rights: *f(ACL)*
retrieve	I	Returns a reference to the EXPRESS data file corresponding to this model schema and optionally downloads that file to the location specified in *localFileRef.* Arguments: in: OPT localFileRef: *BLOB_Ref* out: repositoryRef: *BLOB_Ref* Access rights: *f(ACL)*
Class: *MappingSchema*		
retrieve	I	Same as above, but for the respective mapping specification file. Arguments: in: OPT localFileRef: *BLOB_Ref* out: repositoryRef: *BLOB_Ref* Access rights: *f(ACL)*

V.4 Server operations for communication control

Table V.4 contains a set of operations that are not used for processing the model data themselves, but for controlling the overall client-server communication process (see also Wasserfuhr & Scherer 1999).

Table V.4: Server operations for communication control

Operation	C/I	Description
Class: *CEEsession*		
openSession	C	Opens a session with the project data server. Arguments: in: N/A out: session: *Obj_Ref* Access rights: *f(ACL)*
closeSession	I	Closes all open models and disconnects the client from the server. Arguments: N/A Access rights: *f(ACL)*
execRequest	I	Generic operation to execute a request (see section II.2). Access rights: *f(ACL)*
Class: *Request*		
getRequestStatus	I	Returns the status of an operation executed in asynchronous mode. If the returned status is equal to *finished*, the result of the referenced operation is already available and can be retrieved by a subsequent *getResponse* (see below). Arguments: in: N/A out: requestStatus: *symbol* (one of *acknowledged*, *executing*, *deferred*, *finished*, *failed*) Access rights: *f(ACL)*
getResponse	I	Returns the result of the referenced *Request* object. Arguments: in: N/A out: *Depends on the referenced asynchronous Request in the OID parameter of the operation* Access rights: *f(ACL)*
Class: *MODEL / ModelSchema / MappingSchema / CEEsession*		
upload	I	This operation can be used to *upload* explicitly a data file to the project data repository (normally a side-effect of another operation). Arguments: in: localFileRef: *BLOB_Ref* out: repositoryRef: *BLOB_Ref* Access rights: *f(ACL)*
download	I	This operation can be used to *download* explicitly a data file from the project data repository (normally a side-effect of another operation). Arguments: in: OPT localFileRef: *BLOB_Ref* repositoryRef: *BLOB_Ref* out: N/A Access rights: *f(ACL)*

V.5 Summary of operation types and model states

This table lists all operations presented in the preceding tables V.I to V.4 from the point of view of their meta properties (operation type, model state triggered by the operation, permissible execution modes – see section 4.6). The access rights for the execution of each operation are replicated from these tables for completeness. The presentation sequence is the same as in the above tables.

Table V.5: Operation types and model states

Operation	**Op. Type**	**Triggered model state**	**Permissible exec. modes**	**Access rights**
Generic top-level operations				
Class: *Concept / IfcRoot*				
create	Write	Write	sync	owner
checkIn	CheckIn	CheckIn	sync	owner
checkOut	LongWrite	LongWrite	sync	owner
find	Read	Read	sync/async	f(ACL)
getAccessRights	Read	Read	sync	f(ACL)
getAttribute	Read	Read	sync	f(ACL)
getInstances	Read	Read	sync	f(ACL)
getMethod	Read	Read	sync	f(ACL)
getMethods	Read	Read	sync	f(ACL)
getRelationships	Read	Read	sync	f(ACL)
inspect	Read	Read	sync	f(ACL)
retrieve	Read	Read	sync	f(ACL)
setAccessRights	Write	Write	sync	owner
setAttribute	Write	Write	sync	owner
status	Read	Read	sync	f(ACL)
testAttribute	Read	Read	sync	f(ACL)
unsetAttribute	Write	Write	sync	owner
update	Write	Write	sync	owner
Operations for IfcKernel objects				
Class: *IfcRelationship*				
getRelatingObject	Read	Read	sync	f(ACL)
getRelatedObjects	Read	Read	sync	f(ACL)
getRelations	Read	Read	sync	f(ACL)
Class: *IfcRelGroups*				
addToGroup	Write	Write	sync	owner
removeFromGroup	Write	Write	sync	owner

Table V.5 (cont.): Operation types and model states

Operation	**Op. Type**	**Triggered model state**	**Permissible exec. modes**	**Access rights**
Class: *IfcObject*				
getDocAssignment	Read	Read	sync	f(ACL)
getPropertySets	Read	Read	sync	f(ACL)
getRelations	Read	Read	sync	f(ACL)
Class: *IfcProject*				
merge	Merge	Merge	async	admin
Class: *IfcProduct*				
view	Read	Read	sync/async	f(ACL)
Generic model-level operations				
Class: *MODEL*				
abortTransaction	Abort	Open	sync	f(ACL)
beginTransaction	Control	N/A	sync	f(ACL)
checkIn	CheckIn	CheckIn	sync/async	f(ACL)
checkOut	LongWrite	LongWrite	sync/async	f(ACL)
checkConsistency	Control	N/A	async/sync	f(ACL)
closeModel	Close	Closed	sync	f(ACL)
commitPrepare	CommitPrep.	ReadyToCommit	sync/async	f(ACL)
commit	Commit	Commit	sync	f(ACL)
create	Create	Closed	sync	owner
delete	Delete	Deleted	sync	admin
find	Read	Read	sync/async	f(ACL)
getAccessRights	Read	Read	sync	f(ACL)
getAllModels	Read	Read	sync	f(ACL)
getGroups	Read	Read	sync	f(ACL)
getModelVersions	Read	Read	sync	f(ACL)
getObjects	Read	Read	sync	f(ACL)
getProductObjects	Read	Read	sync	f(ACL)
map	*Source*: Read *Target*: CheckIn	*Source*: Read *Target*: CheckIn	async/sync	f(ACL)
mapTry	Read	Read	async/sync	f(ACL)
match	Control	N/A	async/sync	f(ACL)
merge	Merge	Merge	async	admin
openModel	Open	Open	sync	f(ACL)

Table V.5 (cont.): Operation types and model states

Operation	**Op. Type**	**Triggered model state**	**Permissible exec. modes**	**Access rights**
retrieve	Read LongRead	Read LongRead	sync/async	f(ACL)
rollback	Rollback	Open	async	f(ACL)
setAccessRights	Write	Write	sync	owner
view	Read LongRead	Read LongRead	sync/async	f(ACL)
Class: *ModelSchema*				
getMappings	Read	Read	sync	f(ACL)
getModels	Read	Read	sync	f(ACL)
retrieve	Read	Read	sync	f(ACL)
Class: *MappingSchema*				
retrieve	Read	Read	sync	f(ACL)
Generic operations for communication control				
Class: *CEEsession*				
openSession	Control	N/A	sync	f(ACL)
closeSession	Control	N/A	sync	f(ACL)
execRequest	Depends on the request	Depends on the request	sync/async	f(ACL)
Class: *Request*				
getRequestStatus	Control	N/A	sync	f(ACL)
getResponse	Control	N/A	sync	f(ACL)
Class: *MODEL / ModelSchema / MappingSchema / CEEsession*				
upload	Control	N/A	sync/async	f(ACL)
download	Control	N/A	sync/async	f(ACL)

Appendix VI Parsers and Converters

The proposed representation formats for the conceptual issues addressed in chapters 4-6 of this thesis required the implementation of *parsers* (for all respective syntax specifications), as well as *converters* (to map the data types of one representation to another). This includes the parsing of (1) STEP physical files, (2) Information Container based requests, (3) Knowledge-based expressions and (4) Mapping specification files, and the respective conversion of the data from/to the internal data structures used in the KEE system. As neither of the underlying syntax specifications is of very high complexity, it was possible to implement a straight-forward **top-down recursive parsing** (Sedgewick 1992) in all cases. The parsing and converting of EXPRESS models, which have a much more complicated syntax, has not been part of the work in this thesis.

This appendix provides a short overview of the developed LISP-based parsers and converters for the prototyped project data server PROMISE. They are intentionally implemented as ordinary functions, and not as object-oriented methods, in order to enable their stand-alone usage in other environments. The output is in all cases in the form of *association lists*, with keywords denoting the meaning of the individual sublists. This intermediate meta format was easy to transform into KEE data objects, and should not be difficult to use in another object-oriented LISP environment, such as CLOS (cf. Steele 1990).

VI.1 Parsers

Information Container Parser

This parser accepts as input a string representing a valid InfoContainer externalisation, and produces as output an association list of the form

```
   (INFOCONTAINER label (component-association-lists … ))
```

where for each component sublist, an association list headed by the respective Information Container keyword (AGGREGATION, OREF, BLOB etc.) is built.

Here and in all other functions further below, the optional argument *echo* enables a verbose output which can be helpful by stand-alone use, or for testing purposes. In server mode, *echo* is always automatically set to `false`, i.e. no output.

Function declaration:

```
(defun ptdms-ic-parser (InfoContainer &optional echo) &body )
```

Knowledge-based Expressions Parser

This parser takes as input a knowledge-based expression, specified according to the rules presented in section 4.7, and produces a *Wff* (well-formed formula) suitable for use with the specific KEE inference engine (see Intellicorp 1994). The currently prototyped implementation is only of limited value for other environments, but it could be easily adapted for another rule-based system providing a well-defined input format.

Function declaration:

```
(defun ptdms-kbexpr-parser (kbexpr &optional echo) &body )
```

CSML Parser

This parser accepts as input a CSML mapping specification file, and produces as output an association list in a similar manner as *ptdms-ic-parser*.

Function declaration:

```
(defun ptdms-csml-parser (map-file &optional echo) &body )
```

STEP Physical File Parser

This function is used to parse a STEP physical file defined according to (ISO 10303-21 1994). It produces a list containing the number of elements in the header and the data section of the file, and two arrays for the respective elements in these two sections. Each array element is an association list constructed in a similar way as by *ptdms-ic-parser*, e.g.:

```
   (ENTITY name (attribute-list))
```

where *attribute-list* may contain nested lists depending on the specific values of the attributes. For instance, for an IFC “point” entity, it may have the form:

```
   (ENTITY CartesianPoint ((0.0 1.0 2.0)) )
```

The function does not check if the data are correct w.r.t. the underlying EXPRESS model.

Function declaration:

```
(defun ptdms-step-in (step-file &optional echo) &body )
```

VI.2 Data conversion utilities

STEP Physical File Export

This utility function produces a STEP physical file from the internal representation of a model in KEE. By default, all objects contained in the model are output, but the argument *initial-entities* allows to specify explicitly the classes and/or instances to be processed; in that case, a “partial” STEP file including only the addressed entities is produced.

Function declaration:

```
defun ptdms-step-out (step-file &optional echo
                                &key (floating-pt-precision 12)
                                     (initial-entities nil)) &body )
```

InfoContainer ↔ KEE Value Conversion

This pair of functions enables the conversion of *any* Information Container data type to the respective KEE/LISP data type and vice versa. The additional optional arguments in the second function are only needed for its internal recursive usage, and not by its initial invocation.

Function declarations:

```
(defun ic-to-kee-value (val) &body )
(defun kee-to-ic-value (val &optional kbp (vcmax 1) (vl 1)) &body )
```

STEP ↔ KEE Value Conversion

This pair of functions enables the conversion of *any* STEP physical file data to/from the respective KEE/LISP data types. In the first function, if the argument *step-value* is a STEP file entity reference in the form #nnn, the second argument *inst-refs* *must* be specified to provide the link to the respective KEE object(s). It takes the same form as in the output of the function *ptdms-step-in* presented above.

Function declarations:

```
(defun step-to-kee-values (step-value inst-refs) &body )
(defun kee-to-step-values (kee-unit slot) &body )
```

Appendix VII Referenced Data Models

This appendix provides concise descriptions of the data modelling environments, parts of which have been used or referenced in several of the examples given in this thesis.

In addition, selected data model schemas directly related to these examples are presented. These are: the IFC 2.0 Kernel Model, a proposed (and prototyped) structural domain extension for IFC 2.0, and an application-specific data model for structural analysis with the SOFiSTiK[*)] system.

VII.1 Overview of the STEP Modelling Framework

The scope of the ISO STEP standard is very broad. It covers many industry branches, such as the automotive, process plant, ship building, furniture and electronic industries, along with AEC. Therefore, it does not endeavour to detail how an IT environment based on STEP data models should be implemented, nor to specify how the underlying framework of such an environment should be assembled. These tasks are left to industry sub-committees in STEP, or to developer groups external to STEP. As a consequence, there is actually no clearly defined modelling framework in the standard itself.

On the other side, STEP *does provide* a methodology and an architecture for the construction of standardised data models and *STEP-conformant modelling frameworks*.

The **methodology** is based on a small number of principles, aiming to: (1) define product data models, providing stability and extensibility, (2) support and standardise industry application semantics, (3) specify requirements and implementation forms for product data exchange and sharing, and (4) specify requirements and methods for the assessment of the conformance of STEP-based applications.

The **architecture** fully covers the first two modelling layers presented on fig. 2.5 in section 2.3.2, and provides guidelines for the specification of comprehensive data models as well as constructs facilitating their harmonisation. In particular, this includes:

- definition of the modelling paradigm and the EXPRESS language (ISO 10303-11 1994);
- implementation forms for file-based data exchange (ISO 10303-21 1994), and for data sharing through a standard data access interface (ISO 10303-22 1998);
- an evolving set of standardised data specifications.

The **standardised data models** resulting from the use of the STEP architecture fall into two categories:

1) *Application Protocols* (APs), which are data specifications that satisfy the information needs of a given industry application domain, and
2) *Integrated Resources* (IRs), which are generic data specifications that support the consistent development of application protocols across the industry domains.

*) The SOFiSTiK system is developed and distributed by SOFiSTiK GmbH, Oberschleißheim, Germany.

The approach by which STEP APs are constructed and documented is illustrated on Fig. VII.1 below (cf. Burkett & Yang 1995; Fowler 1995).

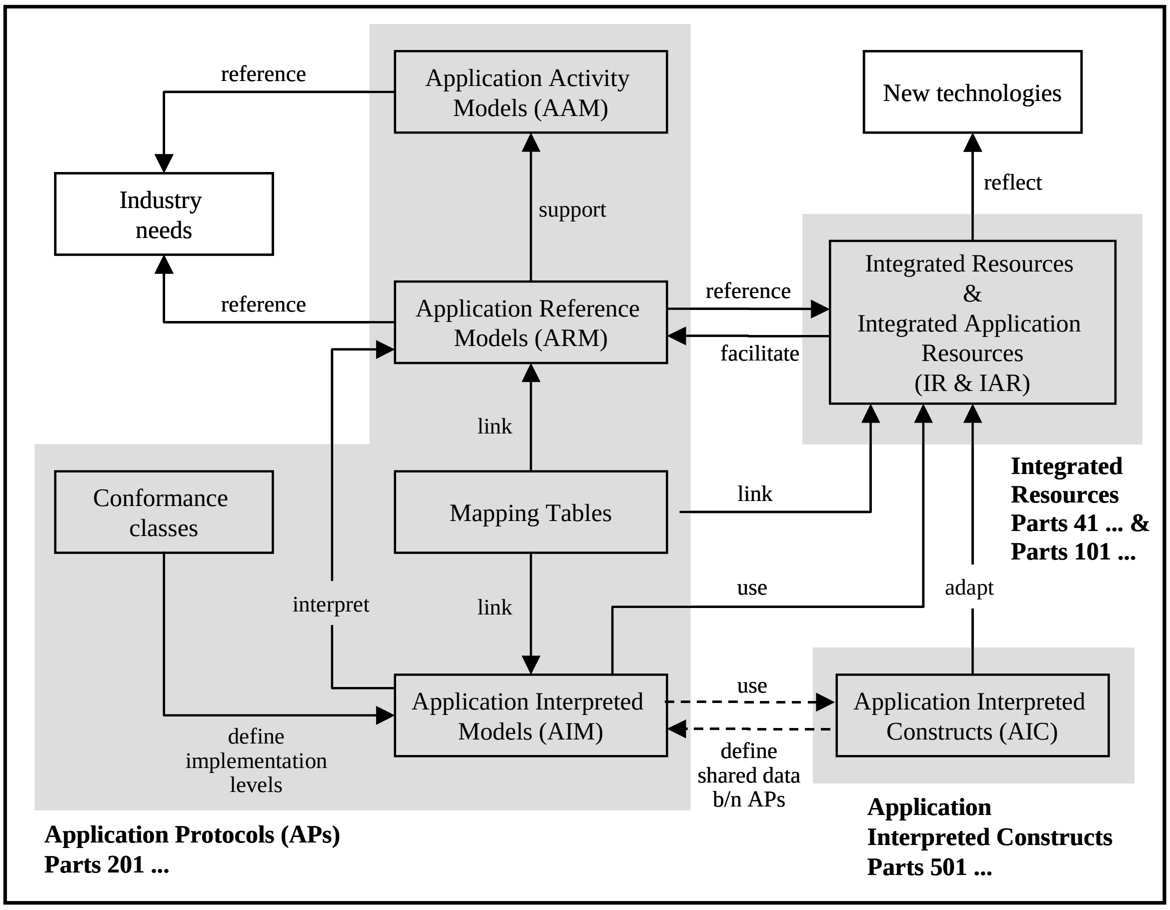

Application Activity Models (AAM)
New technologies
reference
Industry needs
support
reflect
Integrated Resources & Integrated Application Resources (IR & IAR)
reference
Application Reference Models (ARM)
reference
facilitate
link
Conformance classes
Mapping Tables
link
Integrated Resources Parts 41 ... & Parts 101 ...
interpret
link
use
adapt
use
Application Interpreted Models (AIM)
Application Interpreted Constructs (AIC)
define implementation levels
define shared data b/n APs
Application Protocols (APs) Parts 201 ...
Application Interpreted Constructs Parts 501 ...

Fig. VII.1: High-level diagram of the key elements of the STEP architecture
(the arrows express conceptual dependencies, and the grey boxes depict the documentation structure)

Its essence is as follows:

Industry needs are captured by an Application Activity Model (AAM). An Application Reference Model (ARM) provides detailed specifications of the data objects and their relationships that are required to support the activities within the scope of the AAM, and an Application Interpreted Model (AIM) fulfils the requirements that are formally represented by the ARM through selection and constraining of data constructs included in the Integrated Resources. Mapping Tables present the links between the requirements of the ARM and the harmonised specifications included in the AIM. The documentation of all these components, along with respective conformance class specifications, forms a STEP AP.

The Integrated Resources are specified in a set of consistent context-independent schemas. Their purpose appears to be twofold. On one side, they define high-level semantic constructs that belong, by meaning and intention, to the generic product model layer suggested in fig. 2.5. On another side, they contain a number of "primitive" constructs intended to prevent duplicate definitions of basic data types, such as geometrical and topological representation items, materials, date, time, person, organisation, measure, unit etc.

This concept is very interesting, but it is incomplete: the high-level constructs in the IRs have a similar problem as the GARM model (Gielingh 1988a), and the low-level data definitions ensure minimum conceptual redundancy but do not contribute to the overall consistency of a broader framework. What is missing is the "middle part", i.e. a kernel data model.

In anticipation of this problem STEP introduces the so called Integrated Application Resources (IAR) which are supposed to extend the generic resources and fill the gap. Unfortunately, the standard provides only vague guidelines as to how such more specific resources should be constructed. It neither prescribes nor denies that an IAR should play the role of a kernel data model with respect to a set of APs in a given industry domain. As a result, both approaches exist (e.g. Part 101 "Draughting" (ISO 10303-101 1994) is a representative of the first, and the proposed Part 106 "Building Construction Core Model" (ISO 10303-106 1997) is a representative of the second approach, respectively). However, according to the STEP methodology it seems that an AP should actually encompass all elements of the modelling framework for a whole industry domain, including individual model schemas that are harmonised in a similar way to the multi-schema integration approaches known from database research (cf. Reddy et al. 1994; Kim 1995).

For the integration of two or more APs, STEP proposes the use of Application Interpreted Constructs (AICs) that should provide explicit specifications enabling the integration process. However, to the knowledge of the author, a successful implementation of this idea does not exist yet. This issue is in fact related to the semantic interoperability of non homogeneous modelling environments that are more or less neglected by STEP. It was discussed in detail in chapter 5.

VII.2 Overview of the IFC modelling framework

IFC is similar to STEP in many aspects. It adopts major STEP components including the EXPRESS modelling language, the STEP physical file format for data exchange, and a great portion of the Integrated Resources, as well as many of the methodological guidelines of STEP. However, IFC is also different from STEP in some substantial aspects:

1) Whilst STEP assumes a homogeneous model world, the goal of IFC is to define a minimal core model and supporting modelling constructs, such that pre-harmonisation of domain-specific models can be achieved. It foresees also a mapping mechanism to communicate with disperse models surrounding its model parts.
2) Whilst STEP is an ISO standard that spans over all industry sectors, IFC is an industry standard, specifically developed for the needs of the building industry.
3) Unlike STEP, IFC endeavours to provide a complete layered modelling framework, the *IFC Project Model,* as a consistent basis for the development of domain-specific data models corresponding to the disciplines involved in the realisation and maintenance of construction facilities.
4) IFC follows a more pragmatic model development approach than STEP and contains several concepts specifically intended to facilitate faster software implementation.

The IFC model architecture provides a modular structure for the development of model components. As shown on Fig. VII.2 below, it contains four conceptual layers which use a strict referencing hierarchy.

Within each conceptual layer a set of inter-related model schemas are defined.

The ***Resource Layer*** provides resource constructs used by object classes on the higher levels. These classes are largely adopted from the STEP Integrated Resources and are utilised in a similar way. However, high-level generic product representation items are not defined on this layer by IFC, because it provides its own kernel level specifications for that purpose.

The ***Core Layer*** contains the building kernel model and several kernel extensions. The kernel itself provides the basic concepts that are mandatory for all IFC models within the scope of the modelling framework. It determines the overall model structure and decomposition of the framework by defining fundamental concepts concerning relationships, type definitions, attributes and roles, that must be observed in all other layers. The kernel extensions comprise a set of schemas containing AEC specific classes which are all specialisations of the generic classes of the kernel.

The ***Interoperability Layer*** incorporates a set of concepts that are shared by multiple disciplines. Its key idea is the definition of object classes that should serve as "plug-ins" for the classes defined on the domain layer. It is expected that in this way multiple domain models can easily be interlinked.

Finally, the ***Domain Layer*** provides the domain-specific aspect models. Each of these data models is defined in a separate schema which may use or reference any class defined in the Core and the Resource layers. The purpose of each domain schema is the provision of specialised object definitions that are tailored for the use within this domain. Currently, the IFC framework includes domain data models for architectural design, construction management, facilities management and HVAC.

The proposed lean structural domain model, presented in concise form in section VII.5 further below, exposes some of the suggested methods by which the IFC modelling framework is being extended.

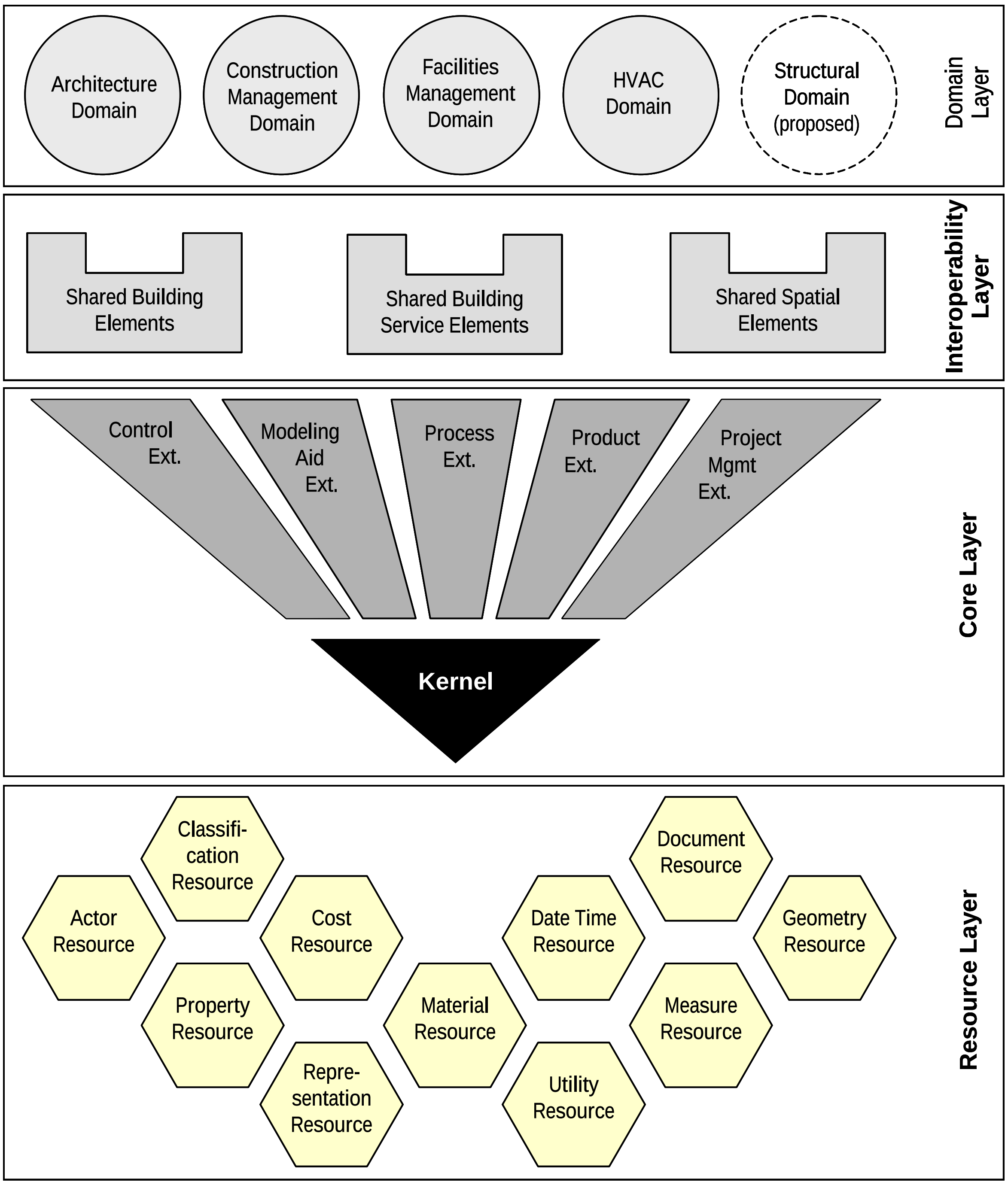


Fig. VII.2: The IFC model architecture / adapted from (IAI 1999c) /

IFC provides a strict methodology for the construction of the model schemas according to this model archutecture. In addition, it defines several constraints to the EXPRESS modelling paradigm that are intended to reduce the complexity of the models and facilitate their implementation by application software.

The most important of these constraint are:

1) the substitution principle of Liskov, which forbids the redeclaration of data types in subclasses, and
2) the specialisation of all object classes by single disjunctive inheritance only (OR-inheritance in EXPRESS terminology).

To enable the integration of independently developed domain data models, a more pragmatic approach compared to the STEP AICs is proposed. Its essence is the definition of generic *Property Objects* which may contain almost all of the IFC data types, defined in dedicated property sets. This “flattens” to some extent the object structure and may lead to redundant and/or ambiguous definitions, because the content of the property objects cannot be controlled automatically by the object system. On the other hand, it is a useful workaround which allows to reduce the overall complexity of the framework and provides a method for user-defined extensions.

VII.3 Overview of the data models in the COMBI project

COMBI was one of the early projects to deal with product data technology research and implementation in the AEC domain. It developed a prototype system for cooperative design focused on the area of structural engineering, but at the same time envisioned a broader environment based on the idea of "integration by communication" (Junge 1991). Though, initially, COMBI had less ambitious goals by comparison to ATLAS, EPISTLE, and later on ToCEE, VEGA and other large projects, it succeeded in establishing a modelling framework that became one of the precursors of the IFC modelling architecture (Junge & Liebich 1998; IAI 1999a).

The COMBI models are arranged in a 3-layer hierarchy, in principle quite similar to the IFC framework, but with less distinct "kernel" level specifications.

(1) The top level of the modelling hierarchy is comprised of the **neutral model layer**. It provides: a) constructs that can be referenced by all other, lower level models, b) generic specifications that can be used as "templates" for the schemas of the domain-specific *partial model layer,* and c) domain-independent definitions of "building" objects that serve for instantiating a "neutral" project context, linking the different modelling perspectives represented by the partial models in the run-time environment. These features of the neutral model layer are supported by three model schemas: `Building_project`, a small "control" model capturing design stages, domains, participants etc., `Building_system`, the "root" schema for all more specific partial model schemas, and `Topology`, supporting the common spatial-topological aspects of the implemented partial (domain) models.

(2) The second, **partial model layer** comprises the schemas defining specific aspects of the addressed design domains. These schemas are similar to the view type models of the ATLAS project and the domain extension models of IFC.

(3) Finally, the third, **application model layer** contains application-specific models, each of which is represented by one schema providing the specification of the application's exchange structures.

A specific feature of the COMBI framework is that its models, and the respective application tools, are only *loosely coupled*. This means that except for the use of common templates, a binding topology, and STEP-like generic resources, the COMBI models are **not tightly harmonised**. Their cooperative use relied heavily on mapping mechanisms (using early versions of CSML and XP-Rule respectively), as well as on the COMBI Communication Manager (CCM), developed by the author of this thesis, which exploited some of the available Internet techniques at that time (mainly ftp-based upload/download, coupled with form-based WWW requests) to provide for coordinated information sharing.

Another specific feature, found only in a few other projects, is the important role allocated to **topology**. The respective COMBI model uses non-manifold topology, as defined in (Weiler 1986), which is in principle very similar to (ISO 10303-42 1994). However, unlike STEP and IFC, this model is not treated as a resource, but as part of the system kernel. This brought certain benefits, but did not awake further interest, probably because of the heavy burden it would have put on end-user CAD systems.

Fig. VII.3 below illustrates the architecture of the COMBI framework, and Table VII.1 on the next page provides summary information about the scope of the modelling effort. The shaded “background pyramid” in Fig. VII.3 is used to make the hierarchical structure of the models more apparent.

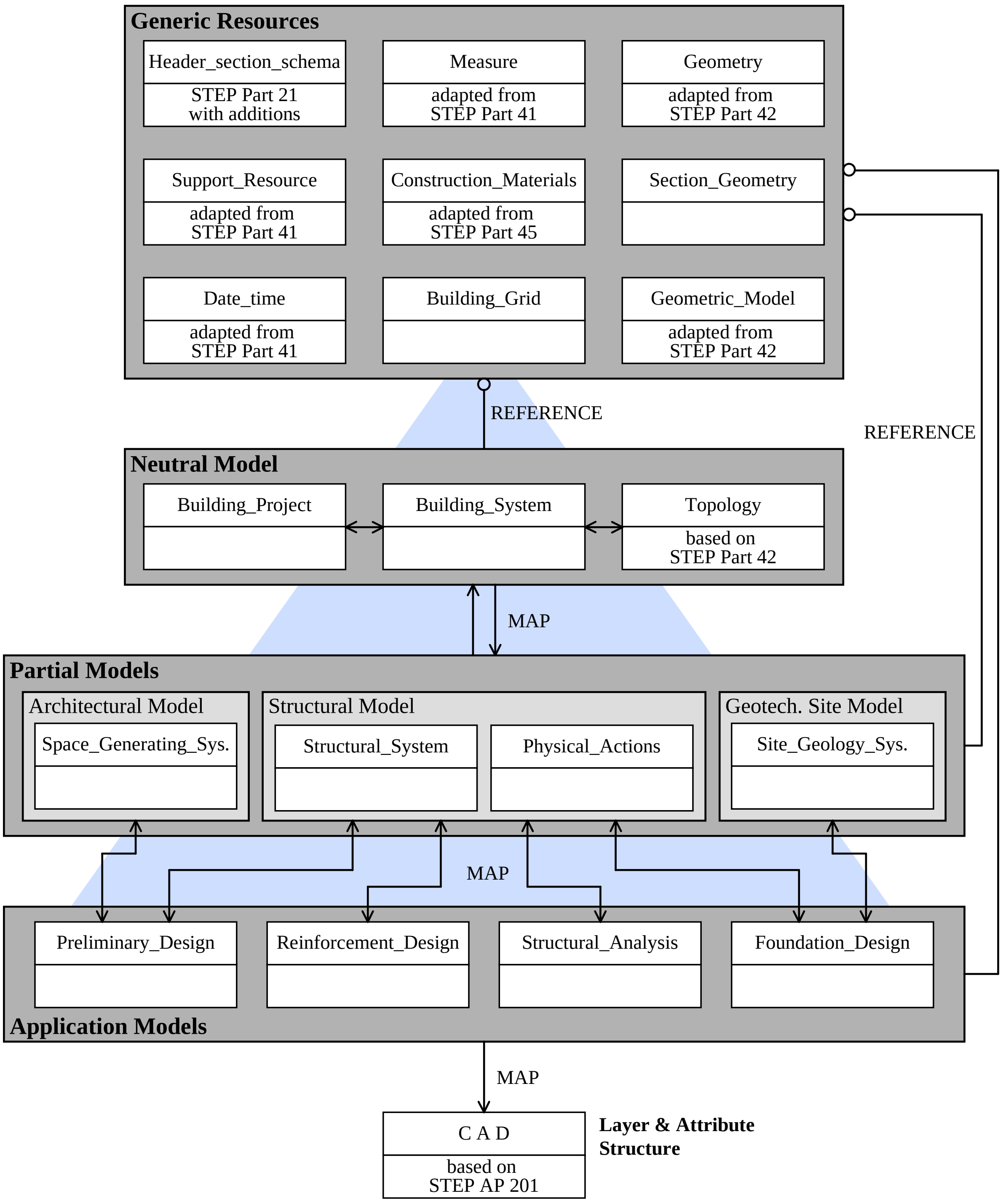


Fig. VII.3:Schema level diagram of the COMBI modelling framework

Table VII.1: Summary of the COMBI model schemas

Category / Layer	Model Schemas	No. of defined entity types	No. of defined data types
Neutral models	Building_project	4	2
	Building_system	17	0
	Topology	24	6
	Subtotal:	45	8
Partial models			
Architectural model	Space_generating_system	9	0
Site model	Site_geology_system	17	3
Structural model	Physical_actions	27	16
	Structural_system	67	11
	Subtotal:	120	30
Application models			
Preliminary design	Preliminary_design	37	7
Foundation design	Foundation_design	14	7
Structural analysis	Structural_analysis	37	7
Reinforcement design	Reinforcement_design	15	5
	Subtotal:	103	26
Generic resources	Header_section_schema	4	5
	Support_resource	2	8
	Date_time	10	11
	Measure	5	24
	Construction_materials	17	11
	Geometry	32	2
	Section_geometry	11	3
	Geometric_model	2	1
	Building_grid	4	0
	Subtotal:	87	65
	Total:	355	129

Note: In the COMBI case studies, not all of these entity and data types have been tested and used.

A comprehensive description of the COMBI models is given in (Ammermann et al. 1994); details of their rationale, implementation and usage are presented in (Katranuschkov & Scherer 1995).

In the development of the models themselves participated several project partners. The main modelling work was done by (in alphabetical order): E. Ammermann, R. Junge, T. Liebich, R. Scherer, and the author of this thesis.

VII.4 Overview of the data models in the ToCEE project

The ToCEE project triggered many of the issues discussed in this thesis, and its modelling framework served as one of the main sources for the validation of the developed concepts.

The ToCEE framework is comprised of a 5-layered set of models (from top to bottom):

(1) The **Meta** model layer presets the basic principles of the whole modelling paradigm. It server for the formal definition of all allowable basic and user-defined data types, as well as for the generic definition of object classes. Following the specifications in the Meta model, each *object class* in any other model has a parent concept and contains a set of attributes describing its state, and a set of operations defining its behaviour. Each *attribute* has in turn a tag indicating if it is part of the EXPRESS entity specification and should thus participate in the data exchange between applications. Each *operation* is defined through its signature (name, type and arguments). In addition, each class definition contains also an identification attribute which can be linked to one of the attributes defined in the EXPRESS schema as is the case with the *ProjectID* attribute of *TC_IfcRoot* and its subclasses.

(2) The **Kernel** model layer defines in three schemas the high-level generic concepts which are common to all lower level models representing product, document and process related information. The first of these schemas, *TC_IfcKernel*, is a modified version of the IFC Kernel Model, version 1.5 final, extended with some additional data management concepts, such as *views*, *approvals*, more comprehensive *access rights* and *authorisation* specifications etc. The second schema, *TC_Communication*, is almost identical to the communication model presented in section 4.3. The third schema, *TC_Population*, is very similar in structure to the SDAI dictionary model (ISO 10303-22 1998), but is extended with important meta model information. It incorporates also the object class *TC_MODEL*, and the respective model-level operations implemented in ToCEE (Hyvärinen et al. 1999).

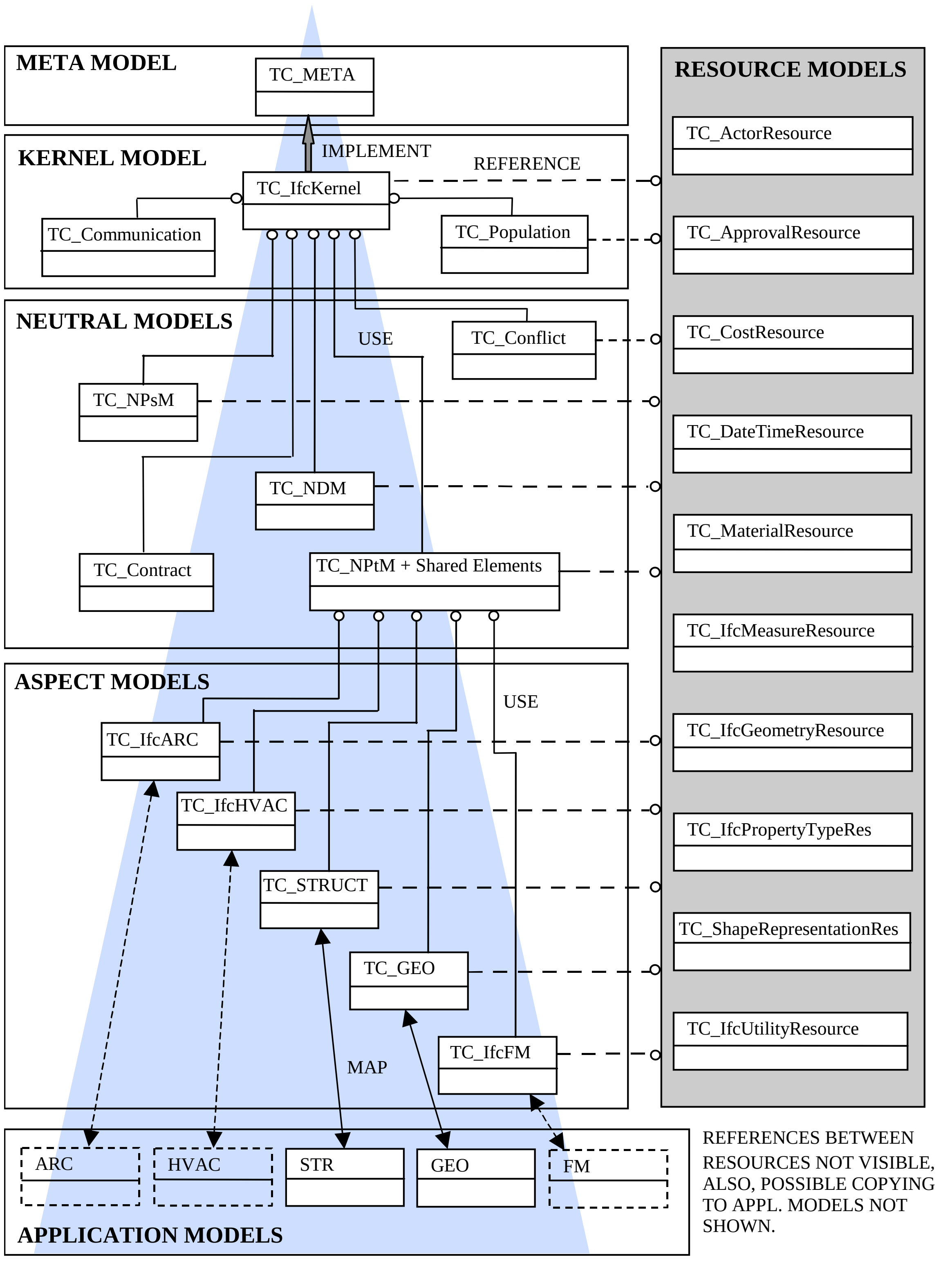
(3) The **Neutral** model layer presents the basic concepts for each modelling perspective, i.e. Neutral Product Model, Neutral Document Model, Neutral Process Model, Neutral Contract Model and a common high-level Conflict Model. These neutral models have been respectively implemented in the different data management servers of the ToCEE environment.

(4) The **Aspect** model layer further specialises the Neutral Product Model for selected domains of building construction by defining aspect models for architectural, structural, HVAC and geotechnical design, as well as for facility management. These models are strictly harmonised with the upper model layers and do not require any mapping transformations. However, they are also quite limited in their scopes.

(5) At last, the **Application** model layer is intended to accommodate the native models of diverse applications. In the scope of ToCEE, this layer includes, as practical examples, only a structural and a small geotechnical application model.

In addition to these five model layers, a set of **independent resources** are available to all other models for referencing - except for the (self-contained) application models, which only copy constructs found useful.

Fig. VII.4 below illustrates the architecture of the ToCEE framework, and Table VII.2 provides summary information giving an impression of its scope.



Note: In the ToCEE scope only a structural and a geotechnical application model have been implemented. The architectural, HVAC and FM tools incorporated in the environment use directly the respective domain (or aspect) models. Ifc in the model name indicates that the model is very close or identical to the respective IFC schema (version 1.5 final).

Fig. VII.4: Schema level diagram of the ToCEE modelling framework

Table VII.2: Summary of the ToCEE model schemas

Category / Layer	Model Schemas	No. of defined entity types	No. of defined data types
Meta model (1)	TC_META	5	11
	Subtotal:	5	11
Kernel models	TC_IfcKernel	21	4
	TC_Communication	11	5
	TC_Population	7	3
	Subtotal:	39	12
Neutral models	TC_PRODUCT_MODEL (TC_NPtM + Shared Elem.)	72	31
	TC_Document_Model (TC_NDM)	25	10
	TC_Process_Model (TC_NPsM)	13	1
	TC_Contract_Model	24	3
	TC_Conflict_Model	26	4
	Subtotal:	160	49
Aspect models (2)			
Architectural model ext.	TC_IfcARC	N/A	N/A
HVAC model ext.	TC_IfcHVAC	N/A	N/A
Structural model ext.	TC_STRUCT	N/A	N/A
Geotechnical model ext.	TC_GEO	N/A	N/A
FM model ext.	TC_IfcFM	N/A	N/A
	Subtotal:	N/A	N/A
Application models (3)			
Struct. design appl. model	SA_APM	127	96
Found. design appl. model	FD_APM	18	8
	Subtotal:	145	104
Resource models	see (IAI 1997) and Fig. VII.4	47	61
	Subtotal:	47	61
	Total:	396	237

Notes:

(1) Formally, **the** Meta model includes 28 entity types and 11 defined data types, providing more strict definition of all basic information items supported in the environment. However, in the implementation, basic types like Integer, Real, Boolean etc. have been directly mapped to the appropriate data types provided by the respective programming languages.

(2) As all aspect models are tightly harmonised with the kernel and neutral models, they introduce no new entity and data types, but only new property definitions, not shown in the above table.

(3) The developed application models cover *only* the data exchange needs of the respective tools used in the ToCEE environment. Unlike the aspect models, they were used as long-form schemas, including all referenced resource objects from the resource model schemas. However, in the ToCEE demonstration scenario, not all of the specified entities have been tested.

A comprehensive description of the ToCEE models and their rationale is provided in (Hyvärinen et al. 1997). An extended summary of the developed concepts and the implementation are presented in (Hyvärinen et al. 1999). The ToCEE product data management services are discussed also in (Hyvärinen et al. 1999), as well as in (Katranuschkov & Hyvärinen 1998) and in (Scherer & Katranuschkov 1999).

Whilst most of the ToCEE models closely follow IFC, version 1.5 final, the framework as a whole introduces several novel concepts, many of which are related to the approach developed in this thesis.

The modelling work itself was basically performed by (in alphabetical order) R. Amor, J. Hyvärinen, P. Katranuschkov, Ž. Turk and R. Wasserfuhr, coordinated by J. Hyvärinen and the author of this thesis.

Fig. VII.5 and Fig. VII.6 below, taken from the GUI of PROMISE, show the inheritance structure of **`TC_PRODUCT_MODEL`**, the basic model schema used in the ToCEE environment. Since this model closely follows the IFC Project Model, the presented figures can be interesting also for analysing the structuring of the IFC framework from the point of view of *class inheritance*, which is not so readily visible with EXPRESS-G.

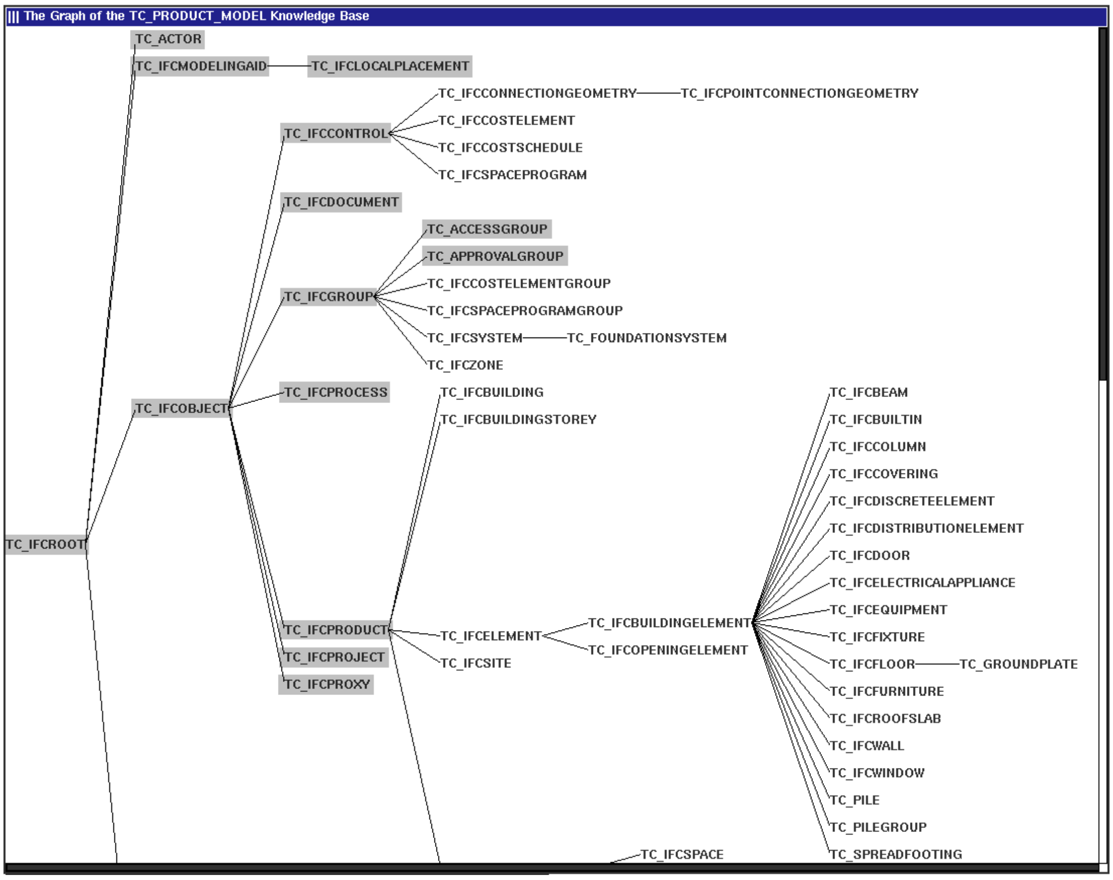


Fig. VII.5: Screenshot (1 of 3) of the inheritance graph of schema TC_PRODUCT_MODEL *(entity classes belonging to the kernel model layer are grey shaded)*

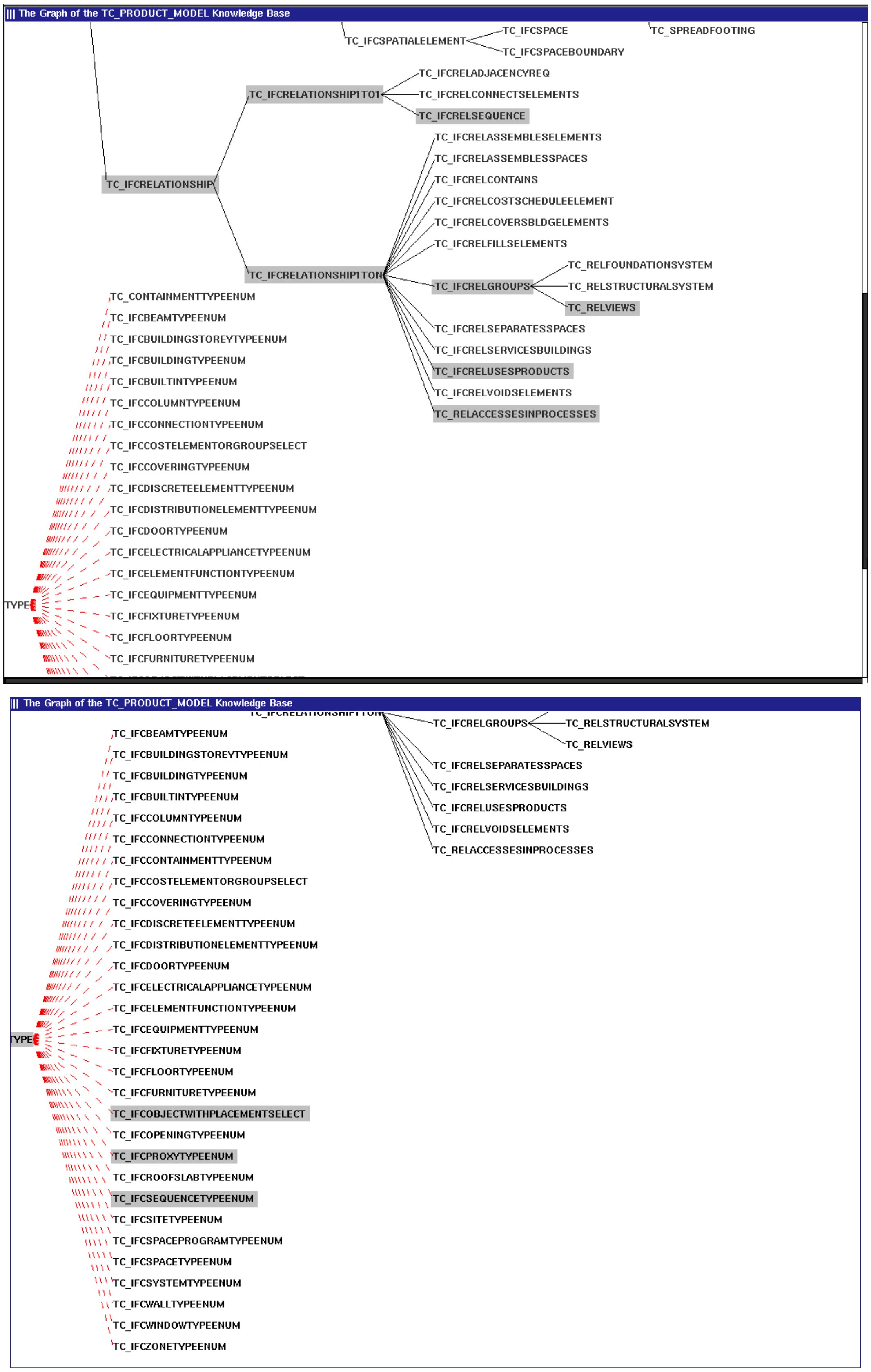
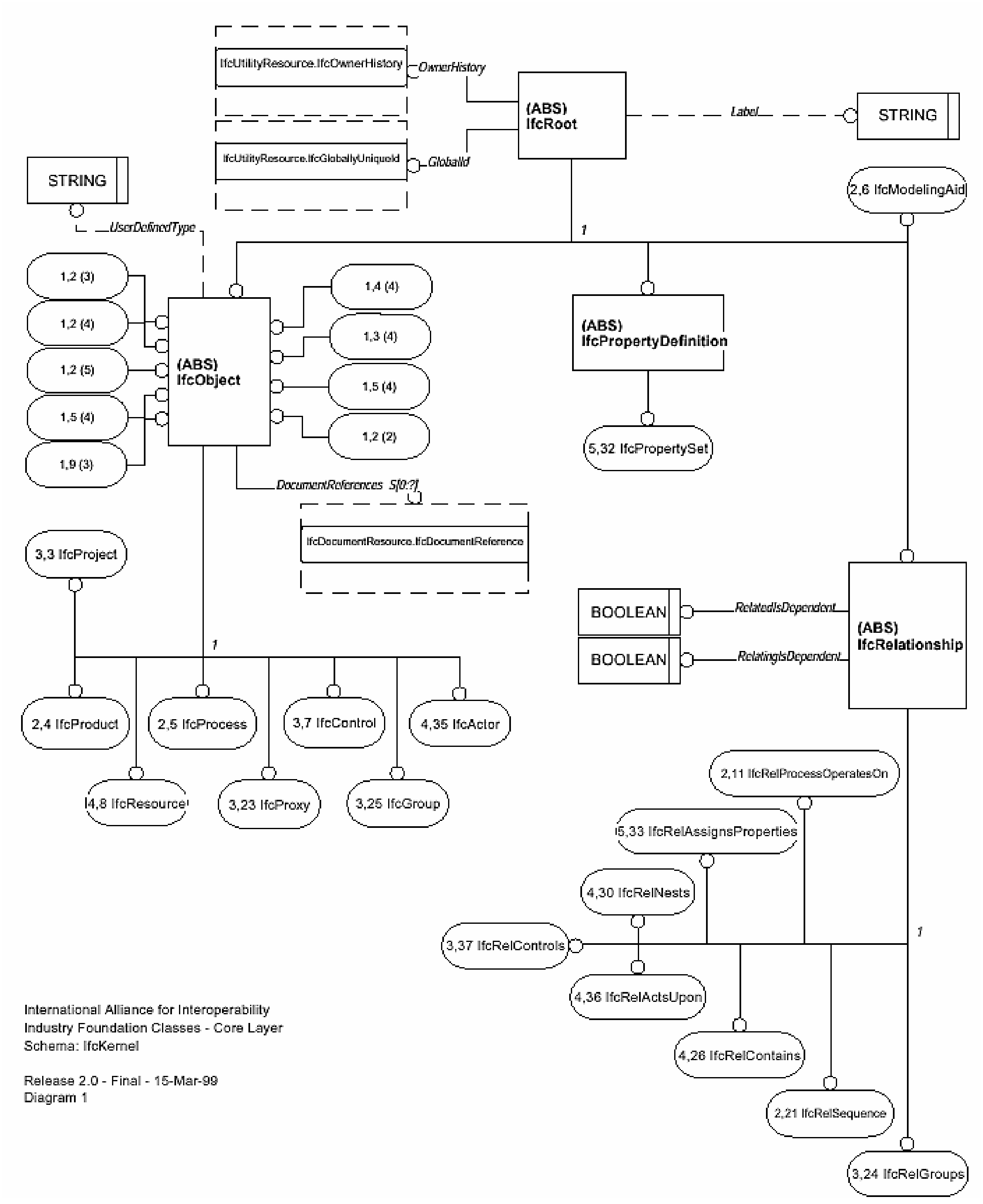


Fig. VII.6: Screenshots (2 & 3 of 3) of the inheritance graph of schema TC_PRODUCT_MODEL

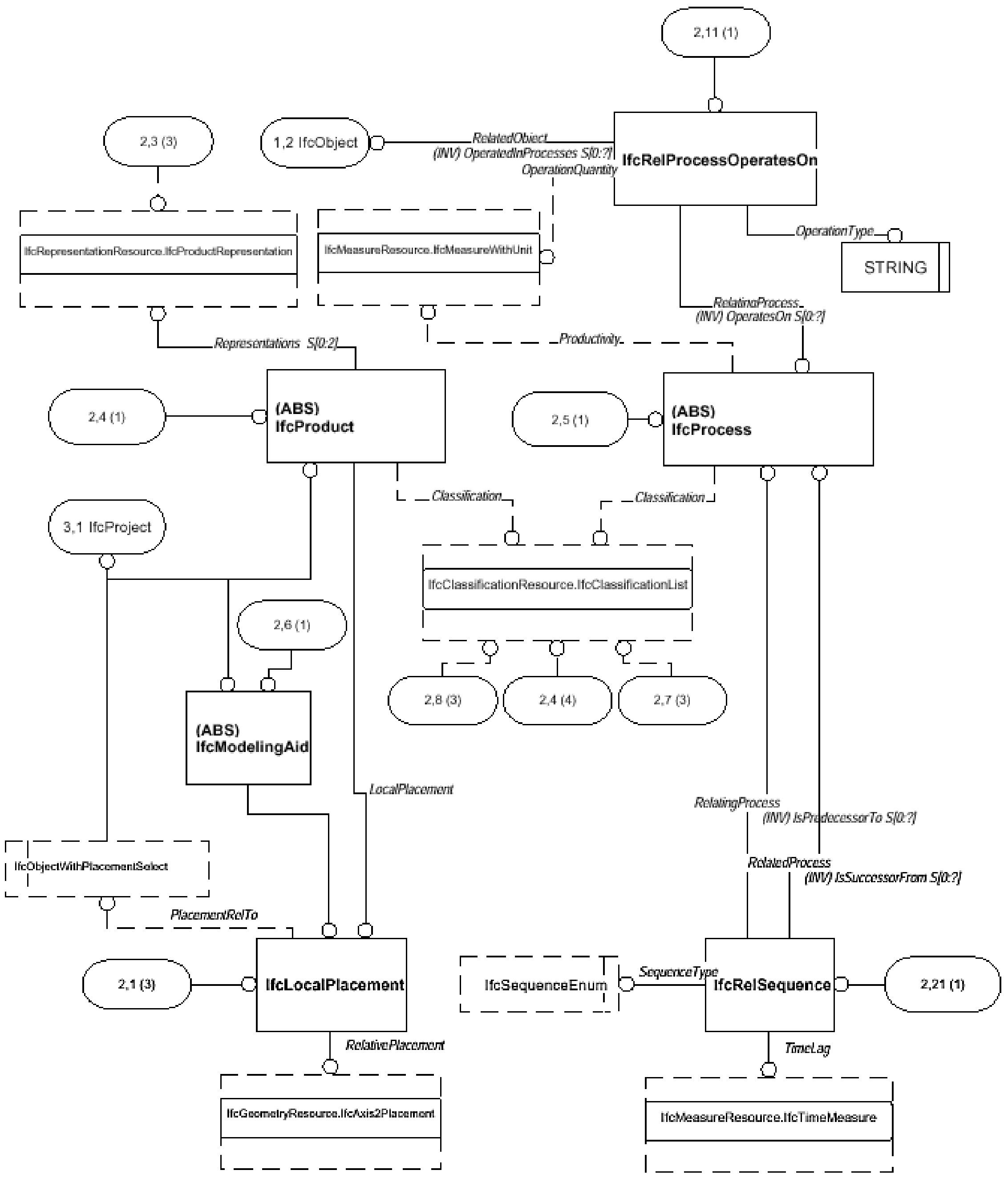
VII.5 Selected Data Model Schemas

VII.5.1. The IFC 2.0 Kernel Model

This section provides an overview of the IFC Kernel Model, release 2.0 final, referenced at many places in this thesis. The EXPRESS-G diagrams, part of the original IFC documentation (cf. IAI 1999c), are reproduced here for *informative* purposes.

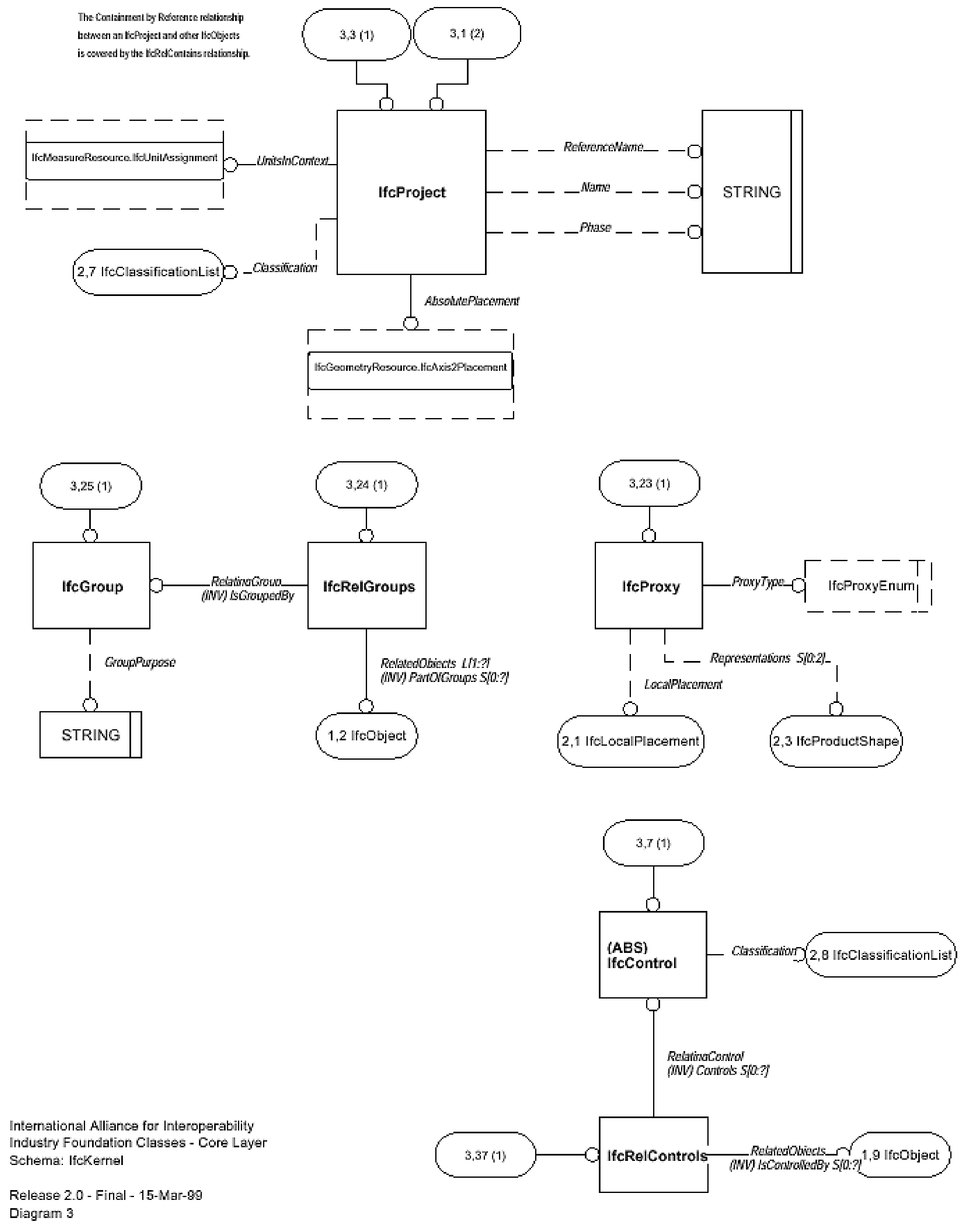


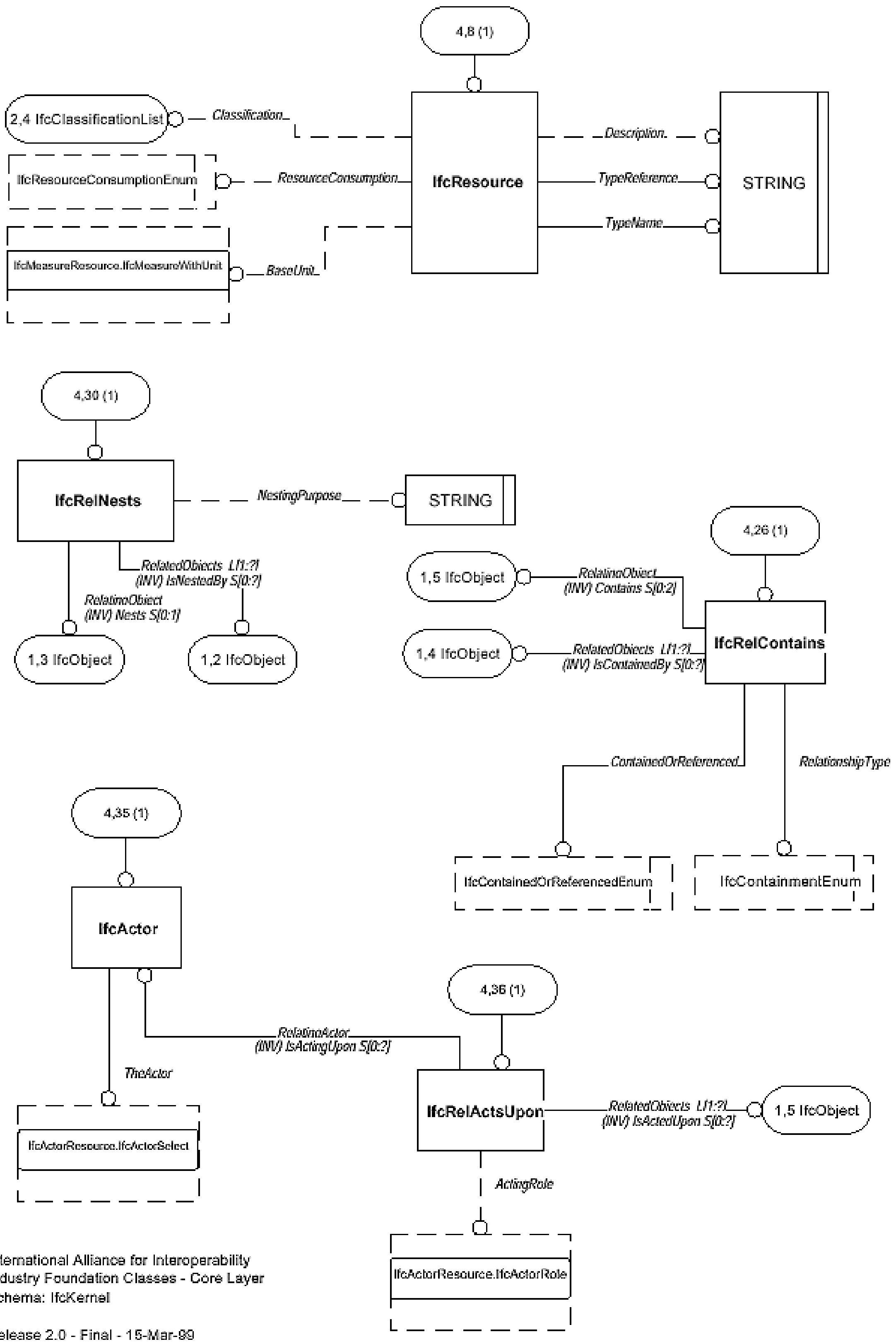
© International Alliance for Interoperability



International Alliance for Interoperability
Industry Foundation Classes - Core Layer
Schema: IfcKernel

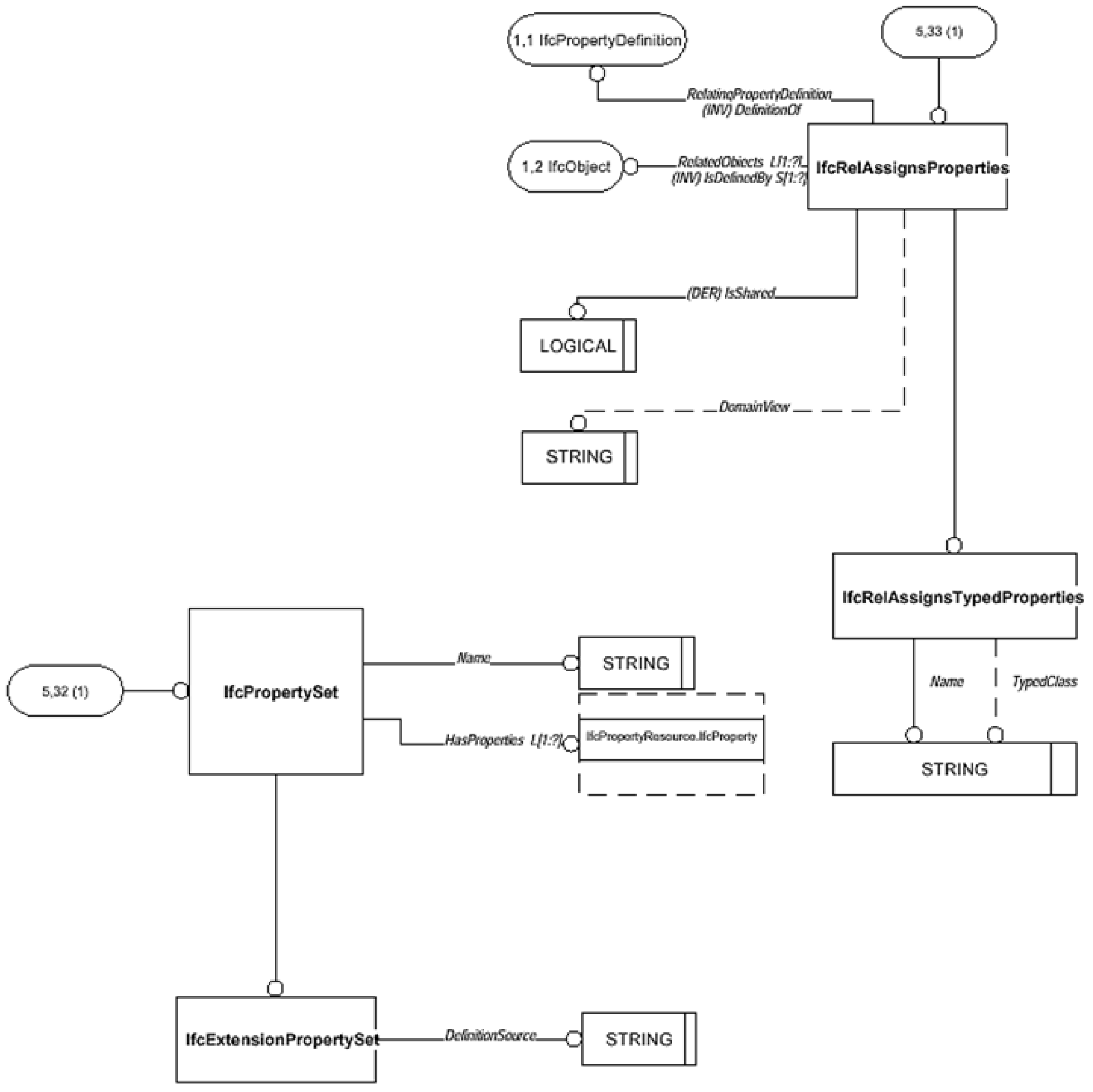
Release 2.0 - Final - 15-Mar-99
Diagram 2


The Containment by Reference relationship
between an IfcProject and other IfcObjects
is covered by the IfcRelContains relationship.
3,3 (1)
3,1 (2)
IfcMeasureResource.IfcUnitAssignment
UnitsInContext
IfcProject
ReferenceName
Name
Phase
STRING
2,7 IfcClassificationList
Classification
AbsolutePlacement
IfcGeometryResource.IfcAxis2Placement
3,25 (1)
3,24 (1)
3,23 (1)
IfcGroup
RelatingGroup
(INV) IsGroupedBy
IfcRelGroups
IfcProxy
ProxyType
IfcProxyEnum
GroupPurpose
STRING
RelatedObjects L[1:?]
(INV) PartOfGroups S[0:?]
1,2 IfcObject
Representations S[0:2]
LocalPlacement
2,1 IfcLocalPlacement
2,3 IfcProductShape
3,7 (1)
(ABS)
IfcControl
Classification
2,8 IfcClassificationList
RelatingControl
(INV) Controls S[0:?]
3,37 (1)
IfcRelControls
RelatedObjects
(INV) IsControlledBy S[0:?]
1,9 IfcObject
International Alliance for Interoperability
Industry Foundation Classes - Core Layer
Schema: IfcKernel
Release 2.0 - Final - 15-Mar-99
Diagram 3



International Alliance for Interoperability
Industry Foundation Classes - Core Layer
Schema: IfcKernel

Release 2.0 - Final - 15-Mar-99
Diagram 4



International Alliance for Interoperability
Industry Foundation Classes - Core Layer
Schema: IfcKernel

Release 2.0 - Final - 15-Mar-99
Diagram 5

<u>Note:</u>

The above diagrams provide only an overview of the entity classes defined in the IFC Kernel Model. For an in-depth view of IFC, the relevant reports and publications (Liebich & Wix 1998; IAI 1999b,c,d), containing details of the rationale, textual descriptions of the separate classes and their purposes, specifications and explanation of property sets etc., should be consulted.

VII.5.2. Prototyped structural domain extension for IFC 2.0

This model has been developed in a diploma thesis (Weise 1999), conducted at the Dresden University of Technology under the supervision and guidance of the author.

The goal of the model was to represent within the frames of the IFC modelling architecture the structural elements, mechanical assumptions and structural analysis models used in the design of a building, so that the structural design intent can be captured, and at the same time, the bilateral connection with the architectural data can be preserved. Not in scope was a detailed description of mechanical models, e.g. as needed for FE analysis, which limited the model in size to the most important structural characteristics that can be used as basis both for the structural designer's work, and to support the co-operative work in a project.

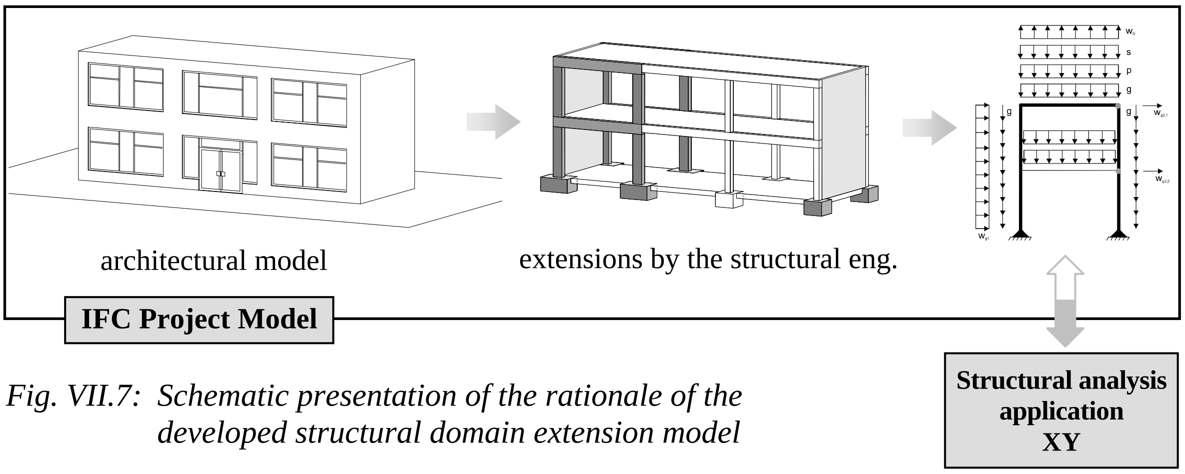

architectural model
extensions by the structural eng.
IFC Project Model
Structural analysis application XY
s
p
g

Fig. VII.7: Schematic presentation of the rationale of the developed structural domain extension model

The developed concept fulfils a preset requirement to use as many as possible of the existing IFC object classes and define as few as possible new "structural domain" objects. This "minimal" extension is reached in first place by utilising the possibilities of *property* objects offered by the IFC model, and in second place by re-arranging the structural information in such way that it fits seamlessly into predefined IFC classes, or in subtyped object structures derived from high-level IFC concepts along the line of the IFC modelling paradigm.

An important aspect in the model design has been the specific treatment of the topological connectivity of structural elements along with the mechanical properties of the connections. Another important aspect for the acceptance of the developed structural domain model are the offered capabilities for the mechanical modelling of a building's bearing structure so that the structural engineer must not follow predefined ways to describe the building structure, but can choose his own preferred way of working. To tackle this issue, both spatial and planar mechanical models, as well as possibilities to define substructures, and support multiple, alternative mechanical models are foreseen. The dependencies between such "partial" mechanical models can be conveniently defined and stored, which normally is not supported in other known similar modelling efforts.

In the context of this thesis, the developed model is interesting in three ways:

(1) It allowed to analyse the benefits, as well as the limitations of the harmonisation approach to IFC model development;

(2) It showed the applicability of a strictly aligned with IFC "lean" structural model to the solution of practical structural analysis tasks, and

(3) It provided the basis for the development of alternative interfaces to a structural analysis system (SOFiSTiK), which was used to perform simple 3-dimensional linear analyses for testing purposes.

The software implementation enabled a quantitative evaluation of some of the interoperability issues addressed in this study. Out of that, the following observations could be made:

1. Because of the tight harmonisation with the IFC Project Model, the generation of the initial structural model was easy, and did not require any mapping specification.
2. In contrast, the interface to an example application-specific model developed for the SOFiSTiK system was, as expected, more difficult to implement.

   A straight-forward implementation in Java, directly using the provided Information Container API presented in Appendix II, yielded - for the example structure shown on Fig. 1.6 (chapter 1) and Fig. VII.7 above respectively - more than 800 client requests which took about 15 min. to execute. Most of this time was spent for "pure" TCP/IP communication, whereas the execution time of the operations at the server took less than half a minute.

   An improved implementation, using knowledge-based templates, allowed to reduce the number of requests by a factor of 3. This decreased the communication overhead almost three times by a slight increase of the server load of about 10-15%.

   Finally, with a mapping specification, only one request is needed, the client interface program can be reduced almost two times in size, and the main load of the needed operations is "pushed" to the server side.

   However, it must be noticed that quantitative results were obtained only for three comparatively small models. Since the relationship between the different approaches is not expected to be linear, more numeric investigations are needed to deduce practical hints when each of these approaches would be best to use.

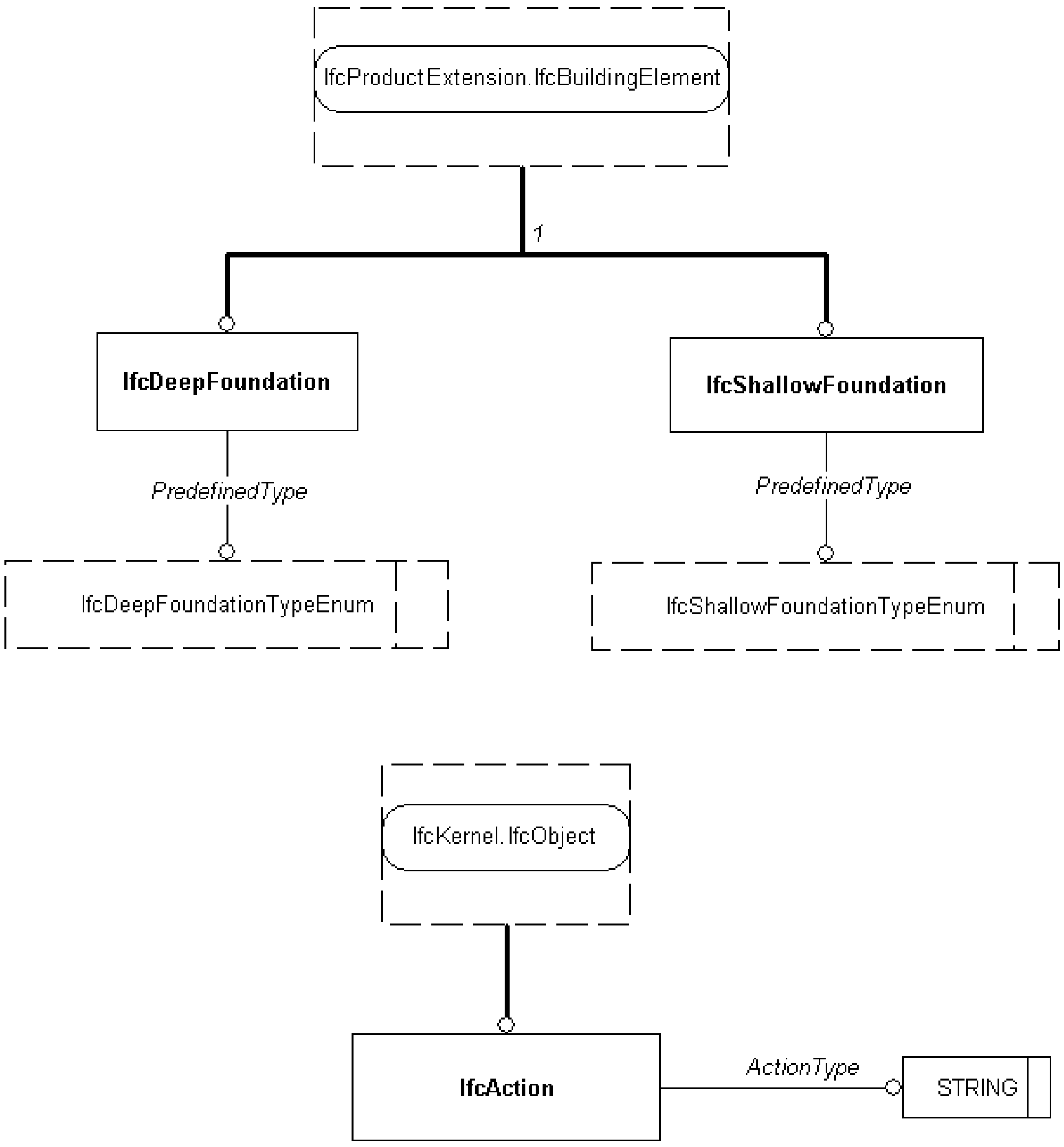
The EXPRESS-G diagrams shown on the next pages provide an overview of the developed model. They include:

(1) an extension of the *IfcSharedBuildingElements* schema on one page, providing two "foundation" entities forgotten in IFC, and an "action" entity seen as useful for other domain models as well, and

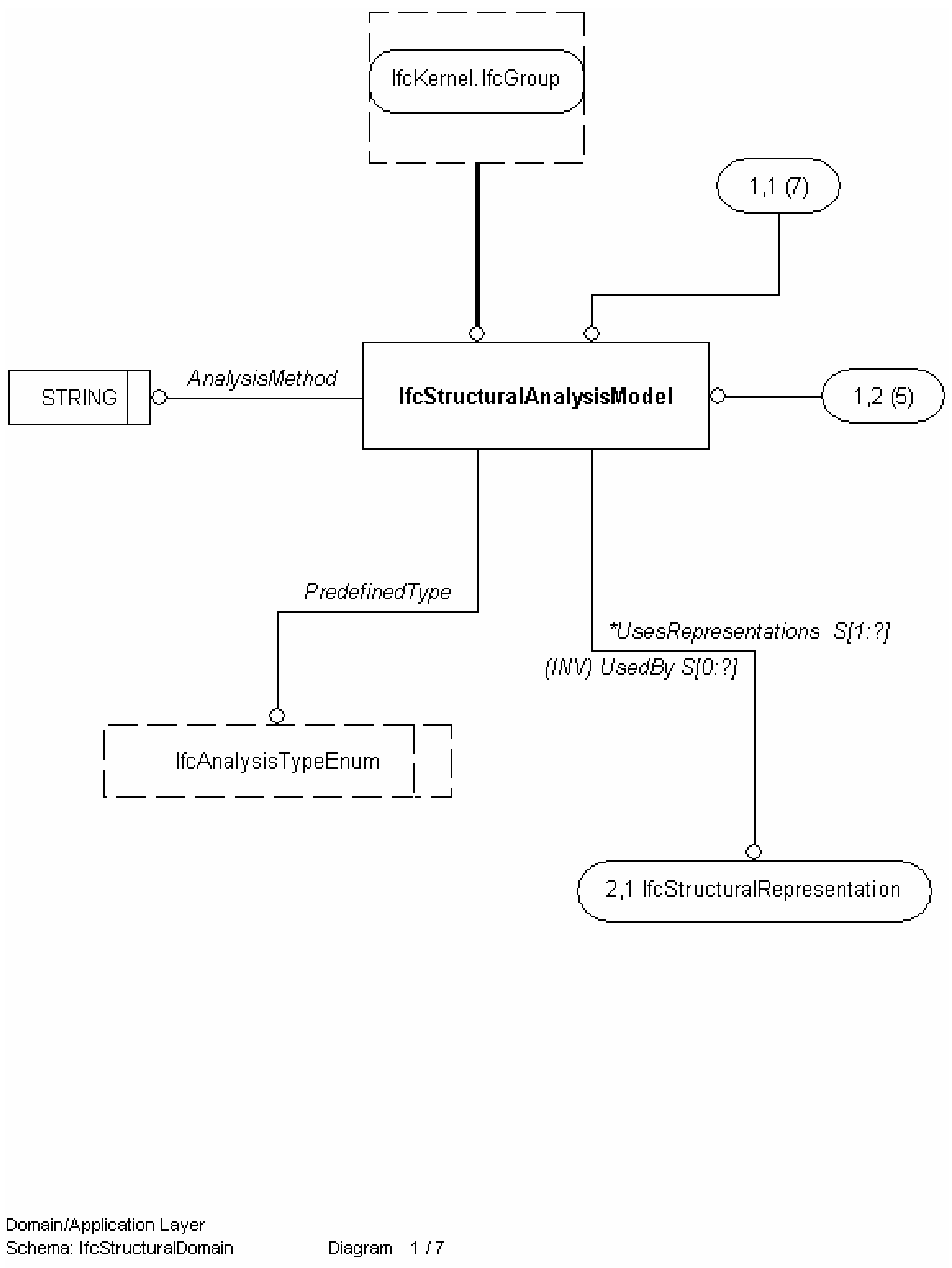
(2) the actual structural domain model specification on seven pages.

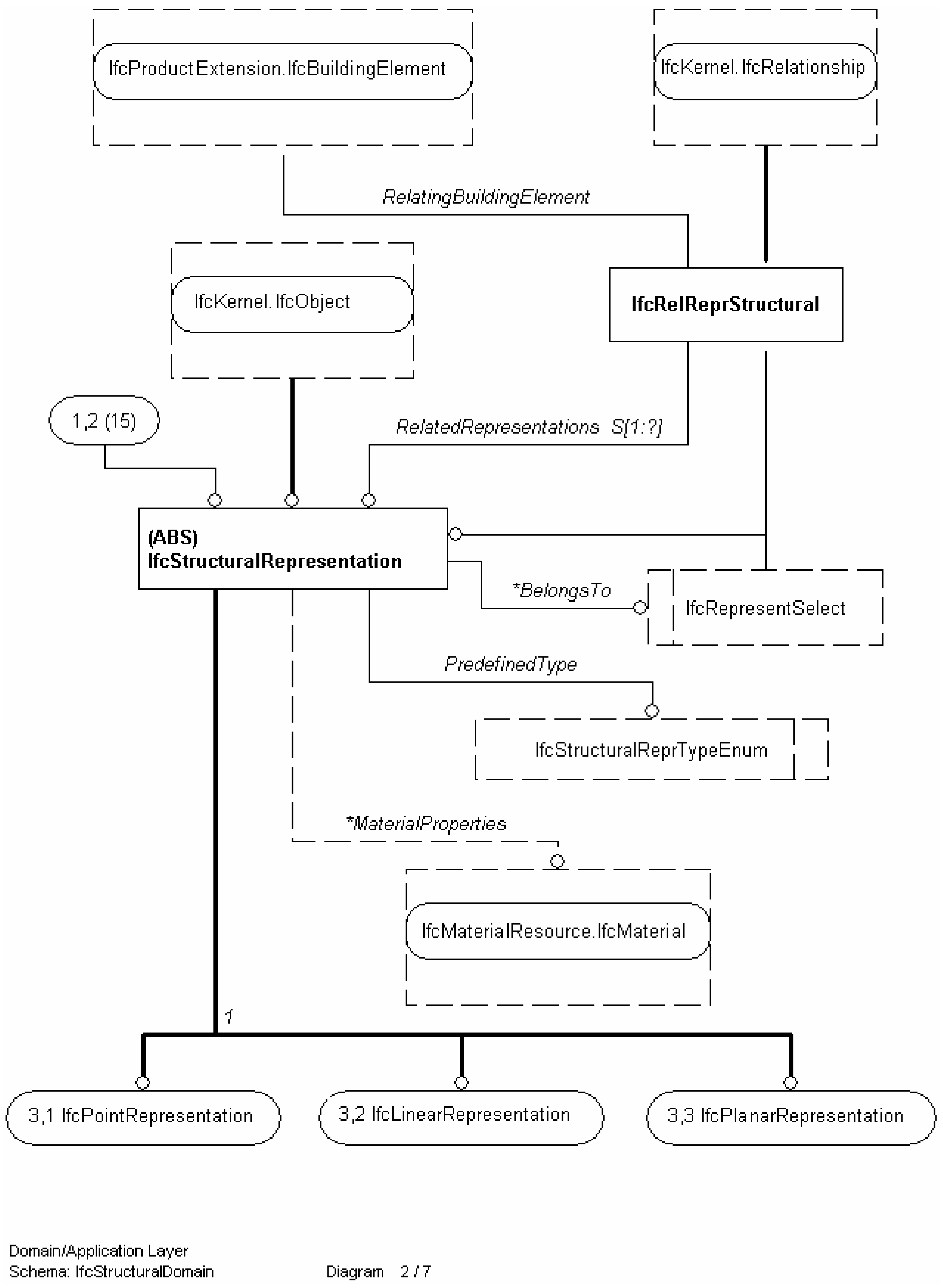
Not shown are the various property objects specified for the model.

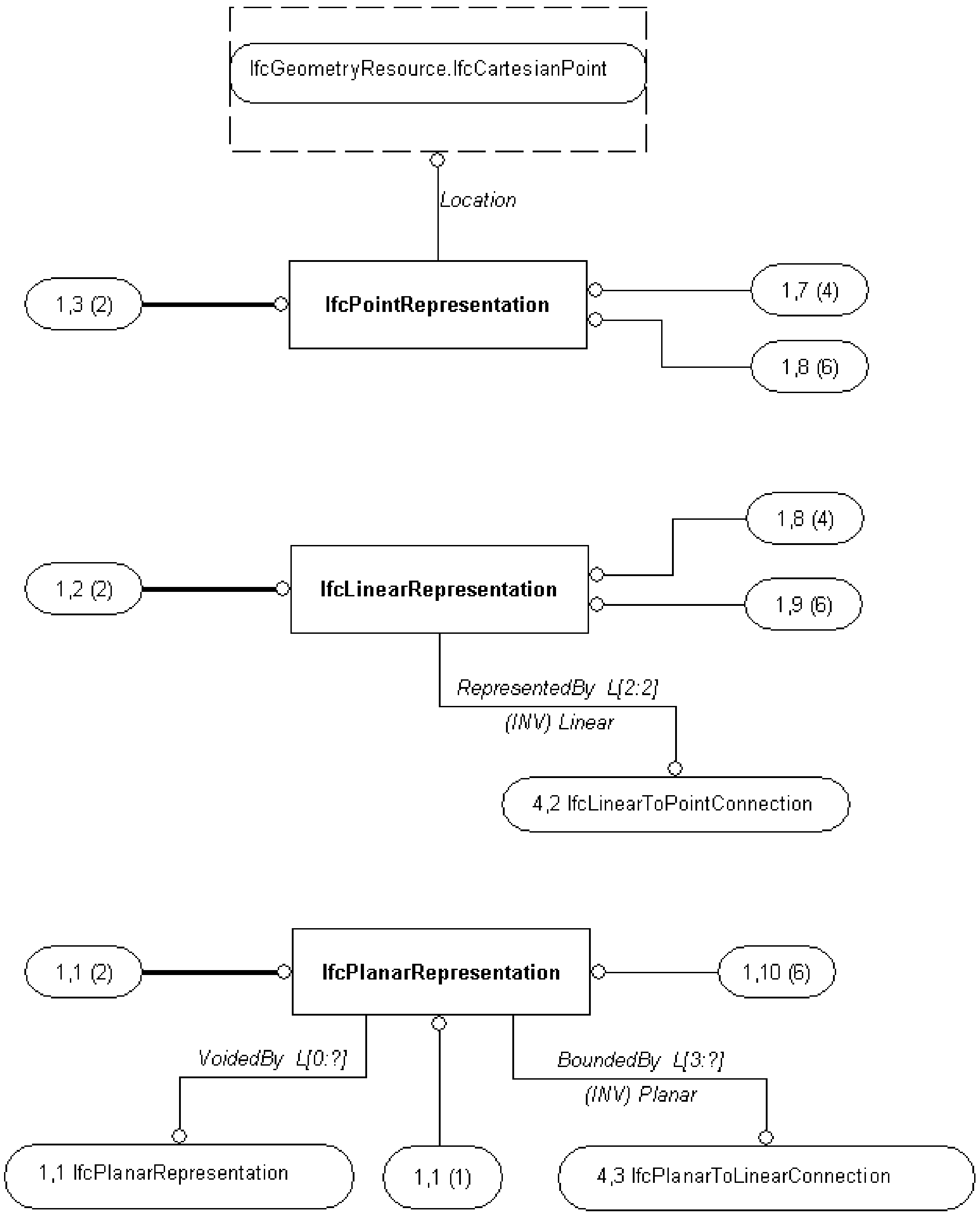
A comprehensive explanation of the model components and the defined 38 property objects is provided in (Weise 1999).



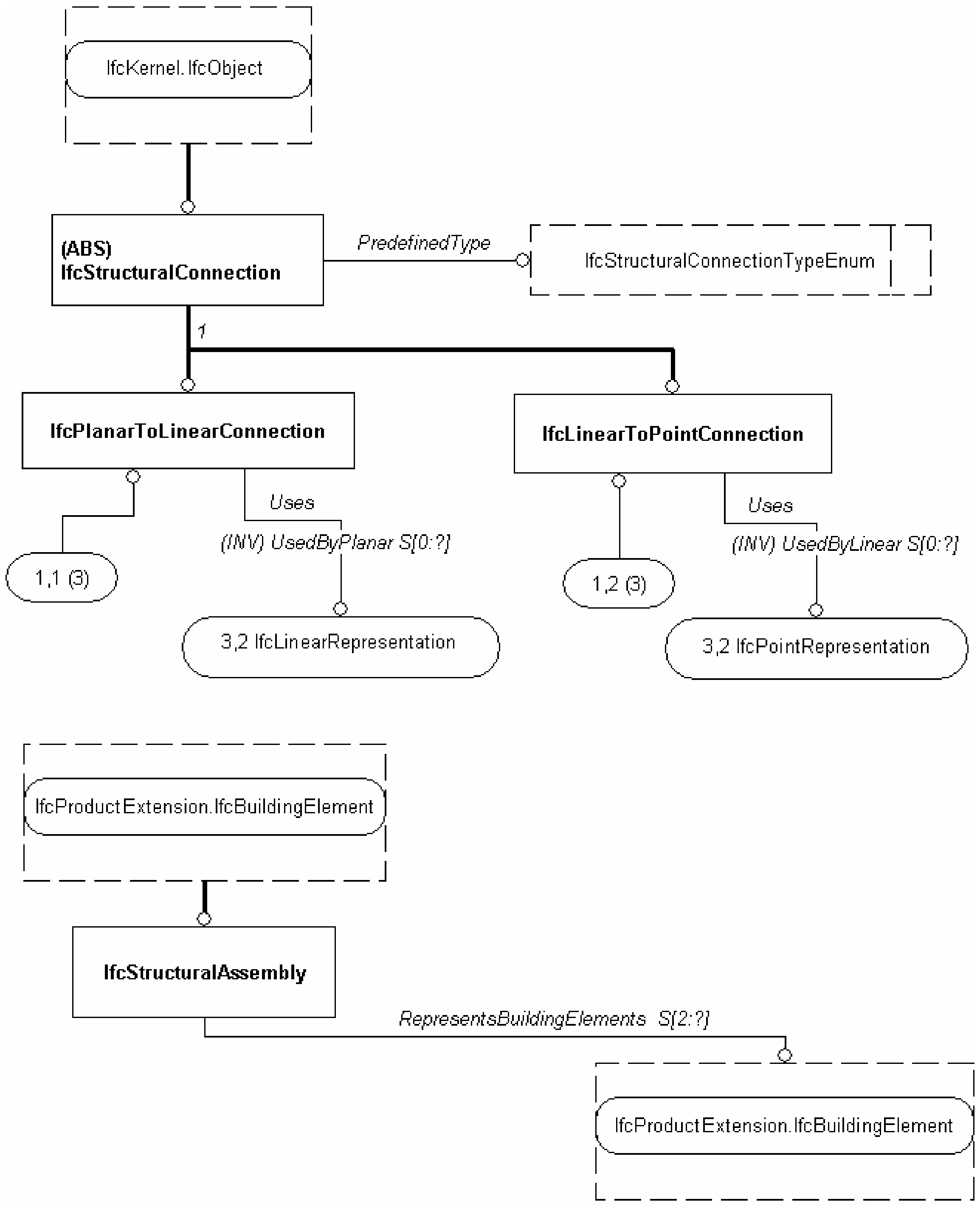
Interoperability Layer Extensions
Ref. Schema: IfcSharedBuildingElements Diagram 1 / 1


IfcKernel. IfcGroup
1,1 (7)
STRING
AnalysisMethod
IfcStructuralAnalysisModel
1,2 (5)
PredefinedType
*UsesRepresentations S[1:?]
(INV) UsedBy S[0:?]
IfcAnalysisTypeEnum
2,1 IfcStructuralRepresentation
Domain/Application Layer
Schema: IfcStructuralDomain
Diagram 1 / 7

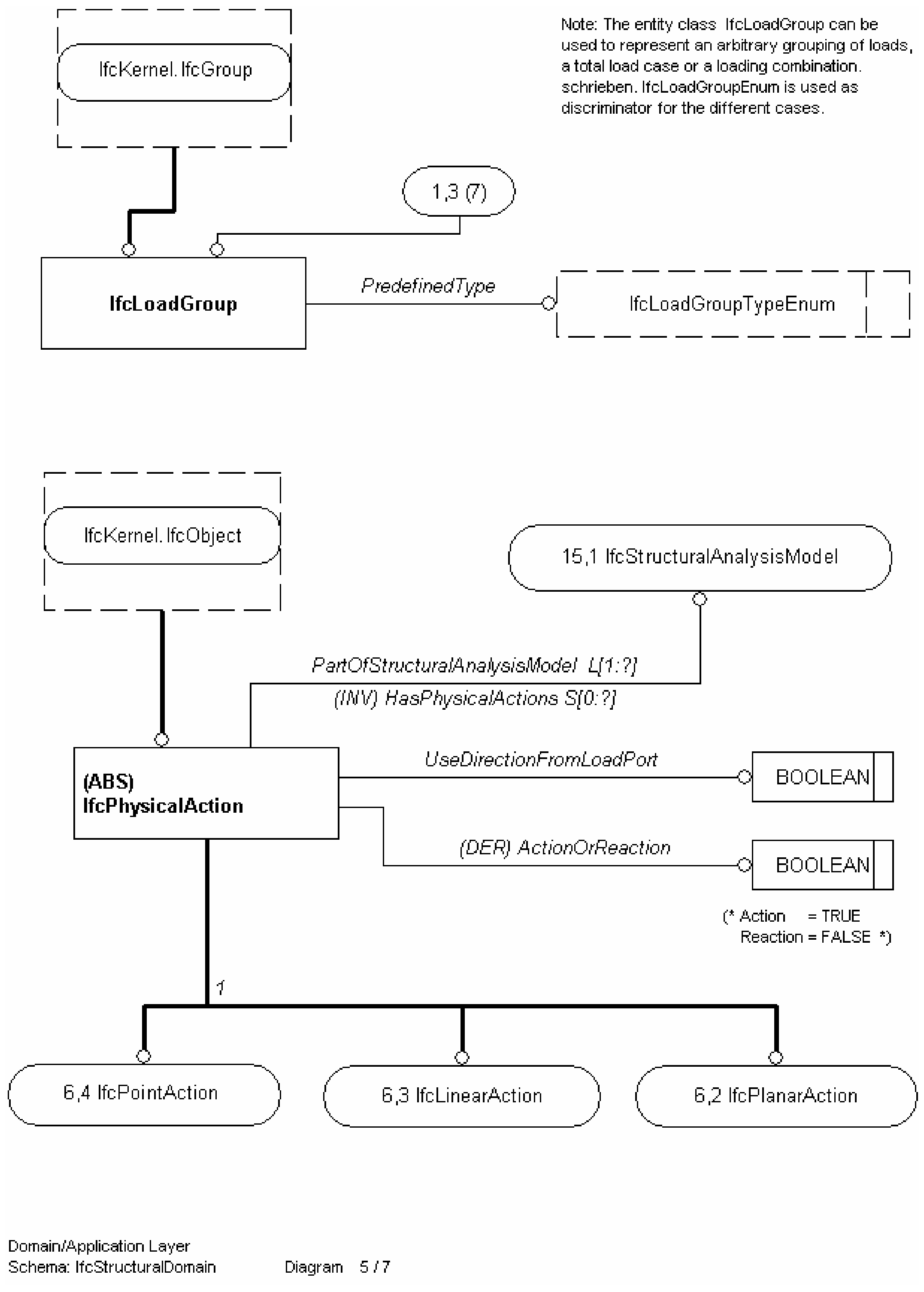

IfcProductExtension.IfcBuildingElement
IfcKernel.IfcRelationship
RelatingBuildingElement
IfcKernel.IfcObject
IfcRelReprStructural
1,2 (15)
RelatedRepresentations S[1:?]
(ABS)
IfcStructuralRepresentation
*BelongsTo
IfcRepresentSelect
PredefinedType
IfcStructuralReprTypeEnum
*MaterialProperties
IfcMaterialResource.IfcMaterial
1
3,1 IfcPointRepresentation
3,2 IfcLinearRepresentation
3,3 IfcPlanarRepresentation
Domain/Application Layer
Schema: IfcStructuralDomain
Diagram 2 / 7

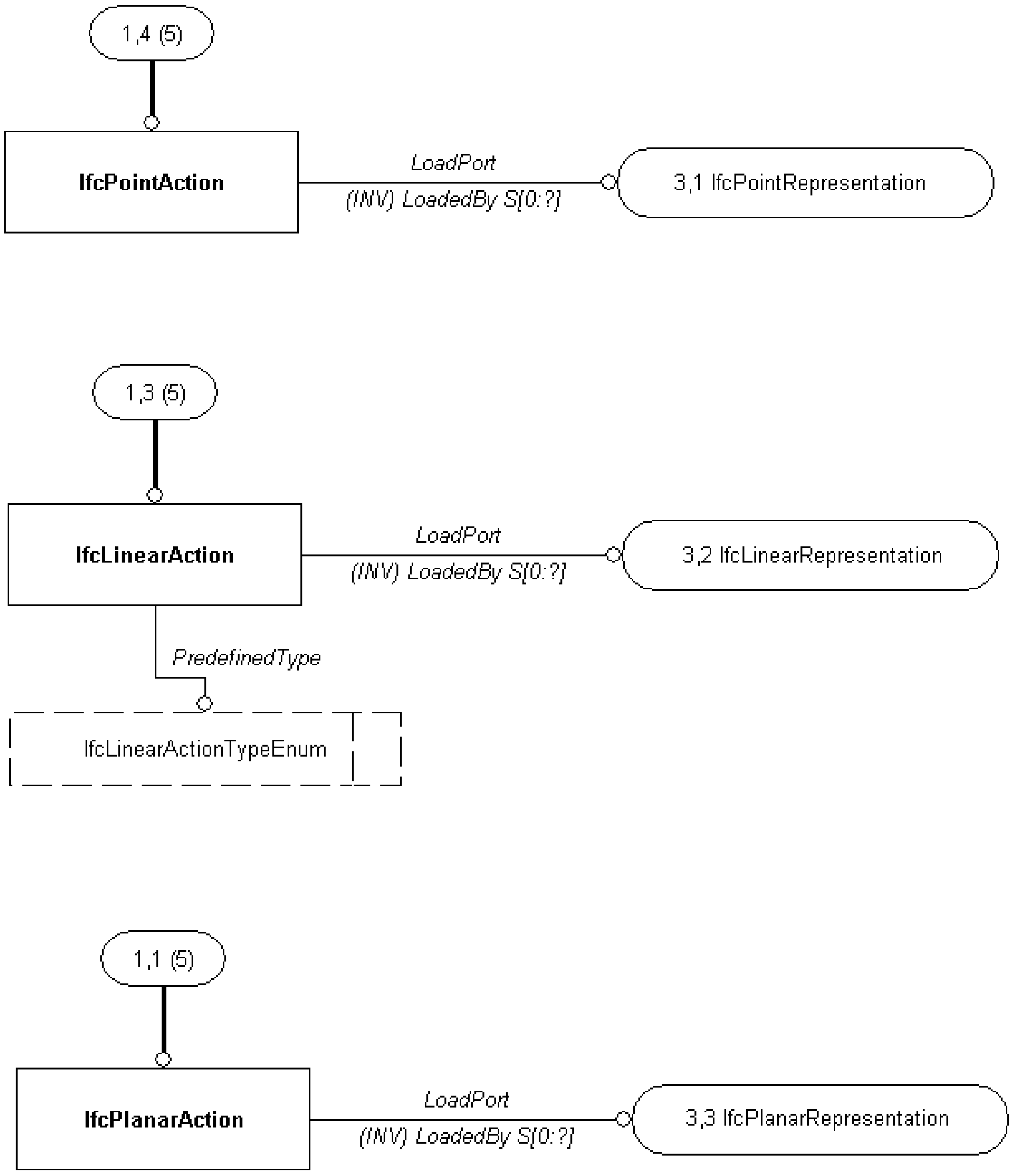


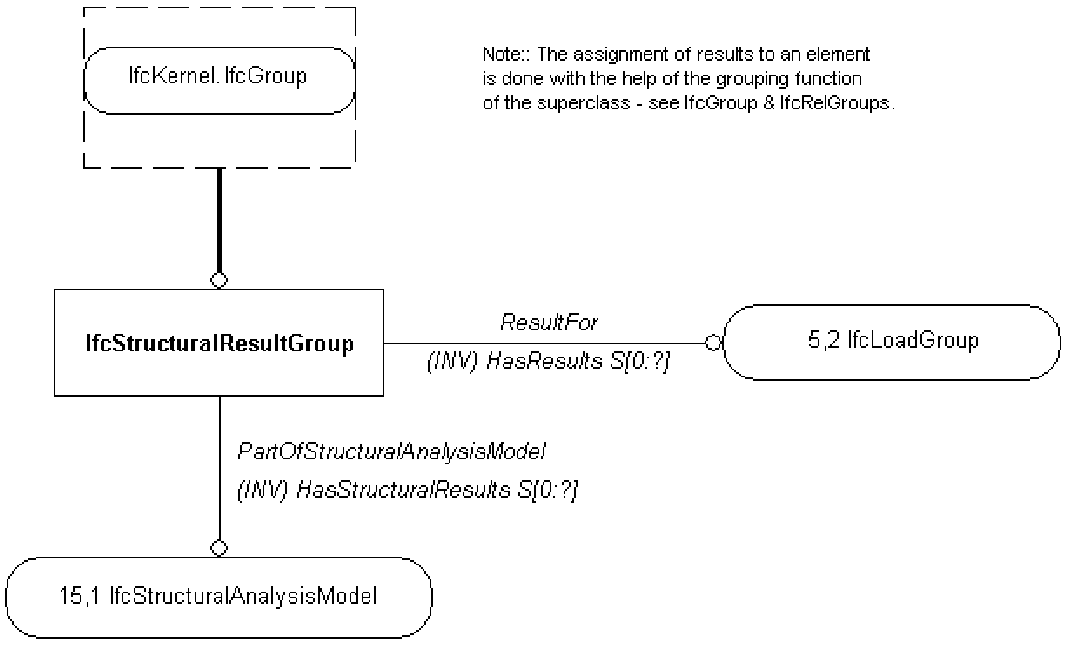
Domain/Application Layer
Schema: IfcStructuralDomain Diagram 3 / 7


IfcKernel.IfcObject
(ABS)
IfcStructuralConnection
PredefinedType
IfcStructuralConnectionTypeEnum
1
IfcPlanarToLinearConnection
IfcLinearToPointConnection
Uses
(INV) UsedByPlanar S[0:?]
1,1 (3)
3,2 IfcLinearRepresentation
Uses
(INV) UsedByLinear S[0:?]
1,2 (3)
3,2 IfcPointRepresentation
IfcProductExtension.IfcBuildingElement
IfcStructuralAssembly
RepresentsBuildingElements S[2:?]
IfcProductExtension.IfcBuildingElement

Domain/Application Layer
Schema: IfcStructuralDomain Diagram 4 / 7

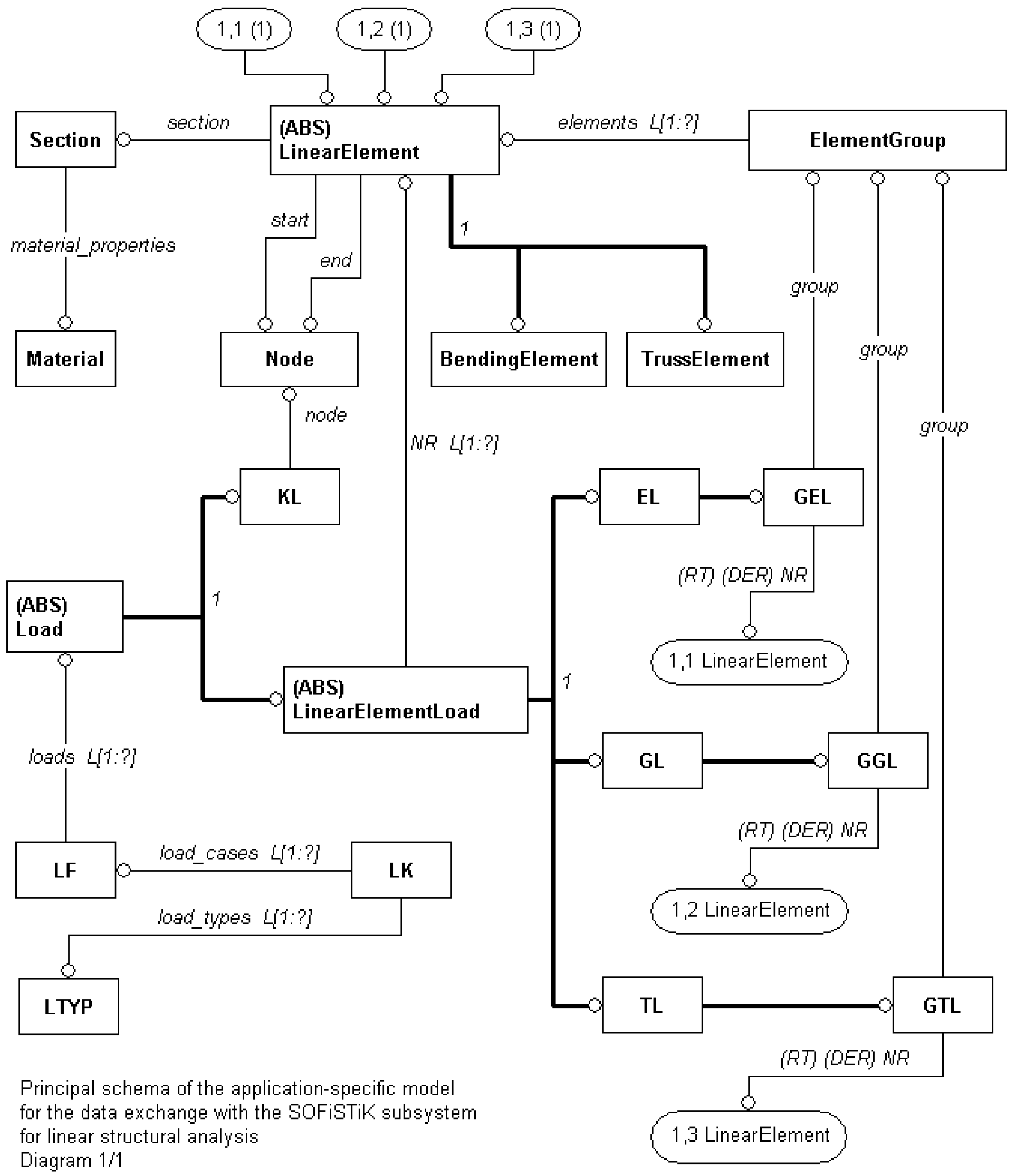

IfcKernel. IfcGroup
Note: The entity class IfcLoadGroup can be used to represent an arbitrary grouping of loads, a total load case or a loading combination. schrieben. IfcLoadGroupEnum is used as discriminator for the different cases.
1,3 (7)
IfcLoadGroup
PredefinedType
IfcLoadGroupTypeEnum
IfcKernel. IfcObject
15,1 IfcStructuralAnalysisModel
PartOfStructuralAnalysisModel L[1:?]
(INV) HasPhysicalActions S[0:?]
(ABS) IfcPhysicalAction
UseDirectionFromLoadPort
BOOLEAN
(DER) ActionOrReaction
BOOLEAN
(* Action = TRUE
Reaction = FALSE *)
1
6,4 IfcPointAction
6,3 IfcLinearAction
6,2 IfcPlanarAction
Domain/Application Layer
Schema: IfcStructuralDomain
Diagram 5 / 7


1,4 (5)
IfcPointAction
LoadPort
(INV) LoadedBy S[0:?]
3,1 IfcPointRepresentation
1,3 (5)
IfcLinearAction
LoadPort
(INV) LoadedBy S[0:?]
3,2 IfcLinearRepresentation
PredefinedType
IfcLinearActionTypeEnum
1,1 (5)
IfcPlanarAction
LoadPort
(INV) LoadedBy S[0:?]
3,3 IfcPlanarRepresentation


IfcKernel.IfcGroup
Note:: The assignment of results to an element
is done with the help of the grouping function
of the superclass - see IfcGroup & IfcRelGroups.
IfcStructuralResultGroup
ResultFor
(INV) HasResults S[0:?]
5,2 IfcLoadGroup
PartOfStructuralAnalysisModel
(INV) HasStructuralResults S[0:?]
15,1 IfcStructuralAnalysisModel

VII.5.3. Example application-specific model for linear structural analysis

This model has been developed "on the fly" (in about two days), for the purposes of testing the model presented in the previous section. It includes only the data required for linear structural analysis by the SOFiSTiK system, but was nevertheless quite useful for the implementation of different versions of the interface to PROMISE, as well as for demonstrating the possible differences between various "structural" models that might need to be tackled in an integration environment. The presented EXPRESS-G schema below is intended only as a brief overview of the model. It shows only the important links between the individual entities, whereas most of the value attributes are skipped. The names of the individual entities are basically taken from the respective keywords of the SOFiSTiK input, the short 2 and 3-letter names denote different loading characteristics.



Principal schema of the application-specific model
for the data exchange with the SOFiSTiK subsystem
for linear structural analysis
Diagram 1/1

References

Amar V., Zarli A., Debras P. & Poyet P. (1997): *Distributing STEP Models with CORBA*, in: Gausemeier J. (ed.) GEN'97 – "International Symposium on Global Engineering Networking", Antwerp, Belgium, 22-24 April, HNI-Verlagsschriftenreihe, Band 21, Paderborn, Germany, pp. 79-96.

Ames A. L., Nadeau D. R. & Moreland J. L. (1996): *Vrml 2.0 Sourcebook* , Second ed., John Wiley & Sons, 654 p.

Ammermann E., Junge R., Katranuschkov P., Liebich Th. & Scherer R. J. (1994): *Concept of an Object-Oriented Product Model for Building Design*, Research Report 1-2/94, Inst. of Applied Computer Science in Civil Engineering, Dresden Univ. of Technology, Germany, 226 p.

Amor R. (1997): *A Generalised Framework for the Design and Construction of Integrated Design Systems*, Ph. D. Thesis, Univ. of Auckland, New Zealand, 350 p.

Amor R., Clift M., Scherer R. J., Katranuschkov P., Turk Ž. & Hannus M. (1997): *A Framework for Concurrent Engineering - ToCEE*, in: Proc. of the European Conf. on Product Data Technology „PDT Days 1997“, 15-16 April, Sophia Antipolis, France, CICA, pp. 15-22.

Amor R. & Hosking J. (1995): *Mappings: The Glue in an Integrated System*, in: Scherer R. J. (ed.) "Product and Process Modelling in the Building Industry", Proc. ECPPM'94, Dresden, 5-7 Oct., Balkema, Rotterdam, The Netherlands, pp. 117-124.

Anderl R. (1995): *Product Data Technology – an Integration Platform for Concurrent and Simultaneous Engineering*, in Faria L. (ed.): Proc. of the International Workshop on Concurrent/Simultaneous Engineering Frameworks and Applications, 5-7 April, Lisbon, Portugal, pp. 41-52.

Angus C. & Dziulka P. (1998): *EPISTLE Framework*, Version 2.0, Issue 1.21, EPISTLE Report *(available from: http://www.stepcom.ucl.ac.uk).*

Assal H. & Eastman C. (1995): *Engineering Database as a Medium for Translation*, in: Proc. CIB W78 - TG10 Workshop on Modeling of Buildings through their Lifecycle, 21-23 August, Stanford University, CA.

Augenbroe G. (1994): *Integrated Building Design Systems in the Context of Product Data Technology*, ASCE Journal of Computing in Civil Engineering, 8(4), pp. 420-435.

Augenbroe G. (1995a): *An Overview of the COMBINE Project*, in: Scherer R. J. (ed.), "Product and Process Modelling in the Building Industry", Proc. ECPPM'94, Dresden, 5-7 Oct., Balkema, Rotterdam, The Netherlands, pp. 547-554.

Augenbroe G. /ed./ (1995b): *COMBINE 2 Final Report*, EU/CEC Joule Programme, Project JOU2-CT92-0196, Directorate Generale XII, Brussels, Belgium.

Bailey I. (1995): *EXPRESS-M Reference Manual*, ISO TC184/SC4/WG5 N243, 93 p.

Batini C., Lenzerini M. & Navathe S. B. (1986): *A Comparative Analysis of Methodologies for Database Schema Integration,* ACM Computing Serveys 18(4), pp. 323-364.

Beucke K. (1995): *Produktmodellierun*g, in: DBV-Workshop “Bauwerkmodelle statt Datenaustausch”, pp. 46-55.

Beucke K. & Ranglack D. (1993): *Computing with Objects: What Does Industry Hope to Gain from It,* in: Cohn L. (ed.) Proc. 5th Int. Conf. on Computing in Civil and Building Engineering (V-ICCCBE), ASCE Publ., NY, pp. 102-109.

Bijnen A. (1995): *Operation Mapping Or How to Get the Right Data?,* in: Scherer R. J. (ed.), “Product and Process Modelling in the Building Industry”, Proc. ECPPM’94, Dresden, 5-7 Oct., Balkema, Rotterdam, The Netherlands, pp. 125-130.

Björk B.-C. (1989): *Basic Structure of a Proposed Building Product Model,* Computer-Aided Design 21(2), pp. 71-78.

Björk B.-C. (1992): *A Unified Approach for Modelling Construction Information,* Building and Environment 27(2), pp. 173-194.

Björk B.-C. (1994): *Conceptual Models of Product, Project and Document Data, Essential Ingredients of CIC,* in: Proc. ASCE 1st Congress on Computing in Civil Engineering, ASCE Publ., NY.

Björk B.-C. (1995): *Requirements and Information Structures for Building Product Data Models*, VTT Publ. No. 245, Espoo, Finland, 87 p.

Björk B.-C. (1999): *Information Technology in Construction: Domain Definition and Research Issues,* Int. J. of Computer Integrated Design and Construction /CIDAC/ 1(1), SETO, London, UK, pp. 3-16.

Böhms M. & Storer G. (1994): *ATLAS - Architecture, methodology and Tools for computer-integrated LArge Scale engineering*, in: Proc. JSPE-IFIP WG 5.3 Workshop, DIISM'93, Tokyo, Japan.

Borgida A. (1991): *Knowledge Representation, Semantic Modeling: Similarities and Differences,* in: Kangassalo H. (ed.): “Entity-Relationship Approach: The Core of Conceptual Modeling”, North-Holland, Amsterdam, The Netherlands.

Borgida A. (1995): *Description Logics in Data Management*, IEEE Transactions on Knowledge and Data Engineering 7(5).

Bralla J. G. (1996): *Design for eXcellence,* McGraw-Hill, New York, NY.

Braun S., Gielingh W., Beekmann D. & Willems P. (1994): *The PISA Product and Process Model,* in: Gausemeier J. (ed.): „CAD´94: Produktdatenmodellierung und Prozessmodellierung als Grundlage neuer CAD-Systeme“, Paderborn, Germany, pp. 149-168.

de Bruijn W., Høyte J., Onneken C. (1995): *The MARITIME AP Factory: A Modelling Environment for Application Protocols*, in: Scherer R. J. (ed.) “Product and Process Modelling in the Building Industry”, Proc. ECPPM’94, Dresden, 5-7 Oct., Balkema, Rotterdam, The Netherlands, pp. 419-427.

Burkett W. C. & Yang Y. (1995): *The STEP Integration Information Architecture*, Engineering with Computers 11(3), pp. 136-144.

Carter D. & Baker B. (1992): *Concurrent Engineering: The Product Development Environment for the 1990s*, Addison-Wesley, NY, 175 p.

Chen P. P. (1976): *The Entity-Relationship Model – Towards a Unified View of Data*, ACM Transactions on Database Systems 1(1), pp. 9-14.

CIRIA (1995): *IT in Construction: Quantifying the Benefits*, CIRIA/CICA Report.

Clark S. N. (1992): *Transformr: A Prototype STEP Exchange File Migration Tool*, National PDES Testbed Report Series, NISTIR 4944, NIST, Gaithersburg, MD.

Cleetus K. J. (1995): *Modeling Evolving Product Data for Concurrent Engineering*, Engineering with Computers 11(3), pp. 167-172.

Codd E. F. (1970): *A Relational Model for Large Shared Data Banks*, Communications of the ACM 13(6), pp. 377-387.

Codd E. F. (1990): *The Relational Model for Database Management,* Second ed., Addison-Wesley, Reading, MA.

Conrad S. (1997): *Föderierte Datenbanksysteme: Konzepte der Datenintegration*, Springer, Berlin-Heidelberg-New York, 331 p.

Cooper K. (1993): *The Rework Cycle: Benchmarks for the Project Manager*, Project Management Journal 14(3), pp. 17-21.

Cragg P. B. & Zinatelli N. (1995): *The Evolution of Information Systems in Small Firms*, Information and Management 29/1995, pp. 1-8.

Crowley A. J. & Watson A. S. (2000): *Concurrent Engineering via PM-based Information Management*, Int. J. of Computer Integrated Design and Construction 2(1), Special Issue on Concurrent Engineering in Construction, SETO, London, UK, pp. 47-53.

Cunis R. (1992): *Das 3-stufige Frame-Repräsentationsschema - eine mehrdimensional modulare Basis für die Entwicklung von Expertensystemkernen*, Ph. D. Thesis, Univ. Hamburg, Dissertationen zur künstliche Intelligenz, Bd. 15, Hundt Druck GmbH, Köln, Germany, 285 p.

Dadam P. (1996): *Verteilte Datenbanken und Client/Server Systeme: Grundlagen, Konzepte und Realisierungsformen*, Springer, Berlin-Heidelberg-New York, 415 p.

Dean E. B. & Unal R. (1992): *Elements of Designing for Cost,* in: Proc. AIAA 1992 Aerospace Design Conf., AIAA-92-1057, Irvine, CA.

Eastman C. (1978): *The Representation of Design Problems and Maintenance of Their Structure,* in: Latcombe (ed.): Application of AI and PR to CAD, North-Holland, Amsterdam, The Netherlands, pp. 335-337.

Eastman C. (1988): *Conceptual Modeling of Buildings. Review,* in: Christiansson P. & Karlsson H. (eds.) "Conceptual Modelling of Buildings", CIB Proceedings Publ. 126, Swedish Building Center, Solna, Sweden.

Eastman C. (1992): *A Data Model Analysis of Modularity and Extensibility in Building Databases*, Building and Environment 27(2), pp. 135-148.

Eastman C. (1993): *Life Cycle Requirements for Building Product Models*, in: Mathur K., Betts M. & Tham K. (eds.) “The Management of Information Technology for Construction”, World Scientific Publishing, Singapore, Aug. 1993, pp. 369-389.

Eastman C., Assal H. & Jeng T.-S. (1995a): *Structure of a Product Database Supporting Model Evolution*, in Proc. CIB W78 - TG10 Workshop on Modeling of Buildings Through Their Life-cycle, 21-23 August, Stanford University, CA.

Eastman C., Chase S. & Assal H. (1993): *System Architecture for Computer Integration of Design and Construction Knowledge*, Automation in Construction 2(2), pp. 95-107.

Eastman C., Jeng T.-S., Assal H., Cho M. & Chase S. (1995b): *EDM-2 Reference Manual*, Tech. Report, Center for Design and Computation, UCLA, Los Angeles, CA, 50 p.

Encarnação J. & Lockemann P. C. (1990): *Engineering Databases: Connecting Islands of Automation Through Databases*, Springer, Berlin-Heidelberg.

EPM (1996): *EDM Interface Programming Guide & Reference Manual, Version 2.0*, EPM Technology, Oslo, Norway.

Fallside D. C. et al. (2000): *XML Schema Parts 0-2*, W3C Candidate Recommendation, 24 Oct. 2000 *(available from: http://www.w3.org./TR/2000/)*.

Fenves S., Hendrickson C., Maher M., Flemming U. & Schmitt G. (1989): *An Integrated Software Environment for Building Design and Construction*, in: Proc. Int. Conf. ARECDAO'89, ITEC, Barcelona, Spain, pp. 71-83.

Fenves S., Rivard H., Gomez N. & Chiou S.-C. (1995): *Conceptual Structural Design in SEED*, J. of Architectural Engineering 1(4), ASCE Publ., NY, pp. 179-186.

Figay N. (1998): *Results of RISESTEP (EP 20459): A User Driven Project to Develop a Platform for Product Data Sharing in a Concurrent Engineering Context*, in: Proc. European Conf. "Product Data Technology Days 1998", 25-26 March, Garston, Watford, UK, published by: QMS, Sandhurst, UK, pp. 133-140.

Fikes R. & Kehler T. (1985): *The Role of Frame-Based Representation in Reasoning*, Communication of the ACM 28(9), pp. 904-920.

Fisher M. & Froese T. (1996): *Examples and Characteristics of Shared Project Models*, J. of Computing in Civil Engineering 10(3), pp. 174-182.

Flemming U. & Woodbury R. (1995): *Software Environment to Support Early Phases in Building Design (SEED): Overview*, J. of Architectural Engineering 1(4), ASCE Publ., NY, pp. 147-152.

Fowler J. (1995): *STEP for Data Management, Exchange and Sharing*, Technology Apprai-sals Ltd., Twickenham, UK, 214 p.

Gardner P. J. (1995a): *IT in Structural Engineering (Results of the Members' Survey on IT and Computers)*, The Structural Engineer 73(21), 7/95.

Gardner P. J. (1995b): *The Infiltration of Information Systems in Engineering Consultancies*, in: Pahl P. J. & Werner H. (eds.) "Computing in Civil and Building Engineering", Proc. 6th Int. Conf. on Computing in Civil and Building Engineering (VI-ICCCBE), Berlin, 12-15 July, Balkema, Rotterdam, The Netherlands, pp. 911-916.

Garrett J. H. Jr., Basten J., Breslin J. & Andersen T. (1989): *An Object-Oriented Model for Building Design and Construction*, in: Nelson J. K. (ed.) "Computer Utilization in Structural Engineering", Proc. ASCE Structures Congress, pp. 332-341.

Garrett J. H. Jr. & Hakim M. M. (1994): *Class-Centered vs. Object-Centered Approaches for Modeling Engineering Design Information,* in: Proc. Int. Colloquium IKM’94, 16-18 March, Weimar, Germany, pp. 267-272.

Genesereth M. & Fikes R. (1992): *Knowledge Interchange Format 3.0, Reference Manual,* Tech. Report Logic-92-1, Computer Science Dept., Stanford University, CA, 49 p.

Gielingh W. (1988a): *General AEC Reference Model,* TNO Report, B1-88-150, Delft, The Netherlands.

Gielingh W. (1988b): *General AEC Reference Model (GARM), An Aid for the Intergration of Applications Specific Data Models,* in: Christiansson P. & Karlsson H. (eds.) "Conceptual Modelling of Buildings", CIB Proceedings Publ. 126, Swedish Building Center, Solna, Sweden, pp. 165-178.Gödel K. (1931): *Über formal unentscheidbare Sätze der Principia mathematica und verwandter Systeme I,* in: Monatshefte für Mathematik und Physik, Jahrgang 37.

Goldfarb C. F. & Prescod P. (1998): *The XML Handbook*, Prentice Hall, NJ.

Gray J. & Reuter A. (1993): *Transaction Processing – Concepts and Techniques*, Morgan Kaufmann Publishers, Los Angeles, CA, 1070 p.

Gruber T. R. (1993): *Towards Principles for the Design of Ontologies Used for Knowledge-Sharing,* in: Guarino N. & Poli R. (eds.): “Formal Ontology in Conceptual Analysis and Knowledge Representation”, Kluwer Academic Publ., Deventer, The Netherlands.

Hakim M. M. (1993): *Modeling Evolving Information About Engineering Design Products: an Object-Centered Approach Combining Description Logic and Object-Oriented Modeling,* Ph. D. Thesis, CMU, Pittsburgh, PA, 150 p.

Hakim M. M., Garrett J. H. Jr. (1994): *Modeling Engineering Design Information: an Object-Centered Approach,* in: Proc. 1st Congress on Computing in Civil Eng, ASCE Publ., NY, pp. 563-571.

Haller H.-W. (1994): *Projektschnittstelle Stahlbau,* Ph. D. Thesis, Institut für Stahlbau und Holzbau, Univ. Stuttgart, Germany, 204 p.

Han C. S., Kunz J. C. & Law K. H. (1999): *Building Design Services in a Distributed Architecture,* J. of Computing in Civil Engineering 13(1), pp. 12-22.

Hannus M., Karstila K. & Serén K.-J. (1995a): *Generic Product Data Model for Product Data Exchange - Requirements, Model and Implementation,* in: Pahl P. J. & Werner H. (eds.) “Computing in Civil and Building Engineering”, Proc. 6th Int. Conf. on Computing in Civil and Building Engineering (VI-ICCCBE), Berlin, 12-15 July, Balkema, Rotterdam, The Netherlands, pp. 283 - 290.

Hannus M., Karstila K. & Tarandi V. (1995b): *Requirements on Standardised Building Product Data Models,* in: Scherer R. J. (ed.), “Product and Process Modelling in the Building Industry”, Proc. ECPPM’94, Dresden, 5-7 Oct., Balkema, Rotterdam, The Netherlands, pp. 43-51.

Hardwick M. (1994): *Towards Integrated Product Databases Using Views,* Rep. 94003, Design and Manufacturing Inst., Rensselaer Polytechnical Institute, Troy, NY.

Hardwick M., Spooner D. L., Rando T. & Morris K. C. (1997): *Data Protocols for the Industrial Virtual Enterprise*, IEEE Journal of Internet Computing, Jan/Feb 97.

Harold E. R. (1997): *Java Network Programming*, O'Reilly & Associates, Sebastopol, CA., 442 p.

Hauser M., Nollau C. & Scherer R. J. (1996): *Intelligent Design Tools as Product Model Interfaces*, in: Turk Ž. (ed.) „Construction on the Information Highway", Proc. CIB-W78 Workshop, Bled 10-12 June, CIB Proceedings Publ. 198, University of Slovenia, Ljubljana, pp. 273-283.

Häußler-Combe U. (1994): *CAD – Aspekte der produktiven Anwendung*, Der Bauingenieur 69(1994), Springer-Verlag, pp. 391-398.

Healy J. & Orr T. L. L. (1996): *Information Technology Use in Consulting Engineering Firms in the Republic of Ireland*, in: Kumar B. & Retik A. (eds.) "Information Representation and Delivery in Civil and Structural Engineering Design", Civil-Comp Press, Edinburgh, UK, pp. 163-170.

Heller P., Roberts S., Seymor P. & McGinn T. (1997): *Java 1.1, Developer's Handbook*, SYBEX Inc., NY.

Hendrickson C. & Au T. (1989): *Project Management for Construction: Fundamental Concepts for Owners, Engineers, Architects and Builders*, Prentice-Hall, Englewood Cliffs, NJ, 537 p.

Herold K. (1997): *Universal Building language*, J. of Computing in Civil Eng. 11(2), pp. 1-4.

Herrmann U. (1991): *Mehrbenutzerkontrolle in Nicht-Standard-Datenbanksystemen*, Informatik-Fachberichte 256, Springer-Verlag, Berlin-Heidelberg, Germany.

van Horssen J. J., Behage B. & Mooij M. (1994): *Conversion*, Internal Report, ESPRIT Project 7280 ATLAS, TNO Delft, The Netherlands.

Hovestadt L. (1993): *A4 - digitales bauen: Ein Modell für die weitgehende Computerunterstützung von Entwurf, Konstruktion und Betrieb von Gebäuden*, Ph. D. Thesis, University Karlsruhe, Germany, 83 p.

Howard R. /ed./ (1996): *Building IT 2005*, Construction IT Forum Res. Report, ICE, Thomas Telford Electronic Publ., London, UK.

Howard H. C., Levitt B. C., Paulson B. C., Pohl J. G. & Tatum C. B. (1989): *Computer Integration: Reducing Fragmentation in the AEC Industry*, J. of Computing in Civil Engineering 3(1), pp. 18-32.

Huovila P., Koskela L. & Lautanala M. (1994*): Fast Or Concurrent - The Art of Getting Construction Improved*, in: Proc. 2nd Int. Workshop on Lean Construction, Santiago, Chile, 28-30 Sept., 11 p.

Hyvärinen J., Katranuschkov P., Hemiö T. & Scherer R. J. (1999): *ToCEE - Product Modelling and Interoperability: Final Report*, EU/CEC ESPRIT Project 20587, Report F4 (public), Inst. of Applied Computer Science in Civil Engineering, Dresden Univ. of Technology, Germany, 78 p.

Hyvärinen J., Katranuschkov P. & Scherer R. J. (1997): *ToCEE - Product Modelling and Interoperability: Concepts for Management Tools / Specification of the Modelling Framework Schemata*, EU/CEC ESPRIT Project 20587, Report F2 / Parts 1 & 2 / (confidential), Inst. of Applied Computer Science in Civil Engineering, Dresden Univ. of Technology, Germany, 241 p.

IAI (1997): *IFC Object Model for AEC Projects, IFC Release 1.5 Model Reference Documentation, Final Version,* IAI Publ. Washington, DC, 174 p.

IAI (1999a): *An Introduction to the International Alliance for Interoperability and the Industry Foundation Classes*, IFC Release 2.0, IAI Publ., Oakton, VA, 21 p.

IAI (1999b): *IFC Object Model Architecture Guide, IFC Release 2.0,* IAI Publ., Oakton, VA, 9 p.

IAI (1999c): *IFC Object Model Guide, IFC Release 2.0,* IAI Publ., Oakton, VA, 93 p.

IAI (1999d): *IFC Software Implementation Guide, IFC Release 2.0,* IAI Publ., Oakton, VA, 57 p.

IAI ST-2 (1998): *IFC R3.0 Domain Project Documentation: [ST-2] Reinforced Concrete Structure and Foundation Structure*, Draft 2, ed. Yasaka A. (project leader), IAI Domain Japan Chapter,Tokyo, Japan.

Intellicorp (1994): *The KEE Software Development System, Version 4.1, User's Manual*, Intellicorp Inc., Mountain View, CA.

ISO 10303-1 IS (1994): *Industrial Automation Systems and Integration -- Product Data Representation and Exchange -- Part 1: Overview and Fundamental Principles,* International Organisation for Standardisation, ISO TC 184/SC4, Geneva.

ISO 10303-11 IS (1994) /Cor.1:1999/: *Industrial Automation Systems and Integration -- Product Data Representation and Exchange -- Part 11: Description Methods: The EXPRESS Language Reference Manual*, International Organisation for Standardisation, ISO TC 184/SC4, Geneva.

ISO 10303-21 IS (1994) /Cor.1:1996/: *Industrial Automation Systems and Integration -- Product Data Representation and Exchange -- Part 21: Implementation Methods: Clear Text Encoding of the Exchange Structure,* International Organisation for Standardisation, ISO TC 184/SC4, Geneva.

ISO 10303-22 IS (1998): *Industrial Automation Systems and Integration -- Product Data Representation and Exchange -- Part 22: Implementation Methods: Standard Data Access Interface*, International Organisation for Standardisation, ISO TC 184/SC4, Geneva.

ISO 10303-28 WD (1999): *Industrial Automation Systems and Integration -- Product Data Representation and Exchange -- Part 28: Implementation Methods: XML Representation for EXPRESS-Driven Data*, ISO TC 184/SC4, NIST, Gaithersburg, MD.

ISO 10303-41 IS (1994) /Cor.1:1999/: *Industrial Automation Systems and Integration -- Product Data Representation and Exchange -- Part 41: Integrated Generic Resources: Fundamentals of Product Description and Support*, International Organisation for Standardisation, ISO TC 184/SC4, Geneva.

ISO 10303-42 IS (1994) /Cor.1-2:1999/: *Industrial Automation Systems and Integration -- Product Data Representation and Exchange -- Part 42: Integrated Generic Resources: Geometric and Topological Representation,* International Organisation for Standardisation, ISO TC 184/SC4, Geneva.

ISO 10303-101 IS (1994) /Cor.1:1999/: *Industrial Automation Systems and Integration -- Product Data Representation and Exchange -- Part 101: Integrated Application Resources: Draughting,* International Organisation for Standardisation, ISO TC 184/SC4, Geneva.

ISO 10303-106 WD (1997): *Industrial Automation Systems and Integration -- Product Data Representation and Exchange -- Part 106: Building Construction Core Model,* ISO/TC184/SC4/WG3/N599, NIST, Gaithersburg, MD.

ISO 10303-201 IS (1994): *Industrial Automation Systems and Integration -- Product Data Representation and Exchange -- Part 201: Application Protocol: Explicit Draughting,* International Organisation for Standardisation, ISO TC 184/SC4, Geneva.

ISO 10303-203 IS (1994) /Cor.1:1996, Cor.2:1998/: *Industrial Automation Systems and Integration -- Product Data Representation and Exchange -- Part 203: Application Protocol: Configuration Controlled Design*, International Organisation for Standardisation, ISO TC 184/SC4, Geneva.

ISO 10303-214 CD (1997): *Industrial Automation Systems and Integration -- Product Data Representation and Exchange -- Part 214: Application Protocol: Core Data for Automotive Mechanical Design Processes*, ISO TC184/SC4/WG3/N578, NIST, Gaithersburg, MD.

ISO 10303-225 IS (1999): *Industrial Automation Systems and Integration -- Product Data Representation and Exchange -- Part 225: Application Protocol: Building Elements Using Explicit Shape Representation,* International Organisation for Standardisation, ISO TC 184/SC4, Geneva.

ISO 10303-227 IS (2000): *Industrial Automation Systems and Integration -- Product Data Representation and Exchange -- Part 227: Application Protocol: Plant Spatial Configuration*, submitted for publication to the International Organisation for Standardisation, ISO TC 184/SC4, Geneva.

ISO 18876-1/-2 WD (2000): *Industrial Automation Systems and Integration – Integration of Industrial Data for Exchange, Access and Sharing (IIDEAS) -- Part 1: Architecture Overview and Description; Part 2: Mapping and Integration Methodology*, International Organisation for Standardisation, ISO TC 184/SC4, Geneva,
(available from: http://www.iso18876.org)

ISO/TC184/SC4/N534 (1997): *Guidelines for Application Interpreted Construct Development*, NIST, Gaithersburgh, MD, 12 p.

ISO/TC184/SC4/WG5/N202 (1994): *PISA Information Modelling Language*.

ISO/TC184/SC4/WG11/N088 (1999): *EXPRESS-X Language Reference Manual*, 72 p.

Jagannathan V., Cleetus K. J., Kannan R., Matsumoto A. S. & Lewis J. W. (1991): *Computer Support for Concurrent Engineering: Four Strategic Initiatives*, Concurrent Engineering, 9/91, pp. 14-30.

Jamsa K., Lalani S. & Weakley S. (1996): *WEB-Programmierung,* Franzis', Feldkirchen, Germany, 580 p. (original title: WEB-Programming, Jamsa Press, Las Vegas).

Junge R. (1991): *Integration by Communication,* in: Proc. 1st International Symposium on Building System's Automation, University of Wisconsin, Madison, WI.

Junge R. & Liebich T. (1998): *Product Modelling Technology – The Foundation of Industry Foundation Classes,* in: Amor R. (ed.) „Product and Process Modelling in the Building Industry", Proc. ECPPM'98, Building Research Establishment, Watford, 19-21 Oct., Clowes Group, Beccles, Suffolk, UK, pp. 259-266.

Junge R., Liebich T. & Ammermann E. (1995a): *Product Modelling for Application,* in: Scherer R. J. (ed.) "Product and Process Modelling in the Building Industry", Proc. ECPPM'94, Dresden, 5-7 Oct., Balkema, Rotterdam, The Netherlands, pp. 111-115.

Junge R., Liebich T. & Ammermann E. (1995b): *Product Modelling for Communication: The COMBI Approach,* in: Pahl P. J. & Werner H. (eds.) "Computing in Civil and Building Engineering", Proc. 6th Int. Conf. on Computing in Civil and Building Engineering (VI-ICCCBE), Berlin, 12-15 July, Balkema, Rotterdam, The Netherlands, pp. 317-322.

Kamara J. M., Anumba C. J. & Evbuomwan N. F. O. (2000): *Developments in the Implementation of Concurrent Engineering in Construction*, Int. J. of Computer Integrated Design and Construction /CIDAC/ 2(1), Special Issue on Concurrent Engineering in Construction, SETO, London, UK, pp. 68-78.

Karstila K., Katranuschkov P. & Mangini M. (1996): *ToCEE - Product Modelling and Interoperability: Requirements,* EU/CEC ESPRIT Project 20587, Report F1 (restricted), Inst. of Applied Computer Science in Civil Engineering, Dresden University of Technology, Germany, 30 p.

Katranuschkov P. (1995): *COMBI: Integrated Product Model,* in: Scherer R. J. (ed.) "Product and Process Modelling in the Building Industry", Proc. ECPPM'94, Dresden, 5-7 Oct., Balkema, Rotterdam, The Netherlands, pp. 511-520.

Katranuschkov P. & Hyvärinen J. (1998): *Product Data Server for Concurrent Engineering in A/E/C,* in: Amor R. (ed.) „Product and Process Modelling in the Building Industry", Proc. ECPPM'98, Building Research Establishment, Watford, 19-21 Oct., Clowes Group, Beccles, Suffolk, UK, pp. 277-289.

Katranuschkov P. & Scherer R. J. (1995): *COMBI - User Manual of the Product Modelling Integration Environment*, EU/CEC ESPRIT Project 6609, Report A4 (public), Inst. of Applied Computer Science in Civil Engineering, Dresden University of Technology, Germany, 68 p.

Katranuschkov P. & Scherer R. J. (1996): *Schema Mapping and Object Matching: A STEP-Based Approach to Engineering Data Management in Open Integration Environments,* in Turk Ž. (ed.) „Construction on the Information Highway" Proc. CIB-W78 Workshop, Bled, 10-12 June, CIB Proceedings Publ. 198, University of Slovenia, Ljubljana, pp 335-346.

Katranuschkov P. & Scherer R. J. (1997): *Framework for Interoperability of Building Product Models in Collaborative Work Environments,* in: Choi C.-K., Yun C.-B. & Kwak H.-G. (eds.) Proc. 7th Int. Conf. on Computing in Civil and Building Engineering, Seoul, Korea, Aug.'97, pp. 627-632.

Katranuschkov P., Scherer R. J., Clift M. & Amor R. (1997a): *ToCEE - Migration Perspectives Towards Concurrent Engineering*, EU/CEC ESPRIT Project 20587, Report J1 (public), Inst. of Applied Computer Science in Civil Engineering, Dresden Univ. of Technology, Germany, 48 p.

Katranuschkov P., Wasserfuhr R. & Scherer R. J. (1997b): *Interoperability of Building Product Models in Collaborative Work Environments*, in: ISO TC184/SC4/WG3/T12 AEC - STEP Building & Construction Group, On-Line Workshop ISFAA-97 „Information Sources for AEC Applications“ (*www.wt.com.au/~ausstep/aec-libraries/*)

Khedro T., Genesereth M. R. & Teicholz P. M. (1994): *Concurrent Engineering Through Interoperable Software Agents*, in: Proc. 1st Conf. on Concurrent Engineering: Research and Applications, Pittsburgh, PA.

Killicote H., Garrett J. H. Jr., Chmielenski T. & Reed K. (1994): *The Context-Oriented Model: An Improved Modeling Approach for Representing and Processing Design Standards*, in: Proc. 1st ASCE Congress on Computing in Civil Engineering, pp. 145-152.

Kim W. /ed./ (1995): *Modern Database Systems: The Object Model, Interoperability, and Beyond*, ACM Press, Addison-Wesley Publ.

Kim W. & Seo J. (1991): *Classifying Schematic and Data Heterogeneity in Multidatabase Systems*, IEEE Computer, 24(12), pp. 12-18.

Kowalczyk W. (1997): *Ein interaktiver Modellierer für evolutionäre Produktmodelle*, Ph. D. Thesis, Berichte aus dem konstruktiven Ingenieurbau 3/97, TU München, Germany.

Krebs T. & Lührsen H. (1995): *STEP Databases as Integration Platform for Concurrent Engineering*, in: Proc. 2nd Int. Conf. on Concurrent Engineering, McLean, Virginia, 23-25 Aug., Concurrent Technologies Corporation, Johnstown, PA, pp. 131-142.

Kühnel R. (1997)*: Die Java 1.1 Fibel*, Addison-Wesley-Longman, Bonn, Germany, 336 p.

Kusiak A. /ed./ (1993): *Concurrent Engineering: Automation, Tools and Techniques*, John Wiley & Sons.

Latham M. (1994): *Constructing the Team*, HMSO, London, UK.

Laufer A. & Cohenca D. (1990): *Factors Affecting Construction Planning Outcomes*, J. of Construction Engineering and Management 116/1, pp. 135-156.

Liebich T., Amor R. & Verhoef M. (1995): *A Survey of Mapping Methods Available Within the Product Modelling Arena*, in: Proc. 5th Int. Conf. of the EXPRESS User Group, Grenoble, 19 p.

Liebich T. & Wix J. (1998): *Highlights of the Development Process of Industry Foundation Classes*, in: Amor R. (ed.) „Product and Process Modelling in the Building Industry“, Proc. ECPPM‘98, Building Research Establishment, Watford, 19-21 Oct., Clowes Group, Beccles, Suffolk, UK, pp. 327-336.

Lockemann P. C., Krüger G. & Krumm H. (1993): *Telekommunikation und Datenerhaltung*, Carl Hanser Verlag, München, Germany, 676 p.

Loffredo D. (1998): *Efficient Database Implementation of EXPRESS Information Models*, Ph.D. Thesis, Rensselaer Polytechnic Institute, Troy, NY, 149 p.

Luiten G., Froese T., Björk B-C., Cooper G., Junge R., Karstila K. & Oxman R. (1993): *An Information Reference Model for Architecture, Engineering and Construction*, in: Mathur K. S., Betts M. P. & Tham K. W. (eds.) "The Management of Information Technology for Construction", World Scientific Publishing, Singapore.

Minsky M. (1975): *A Framework for Representing Knowledge*, in: Winston P. H. (ed.) "Computer Science Series: The Psychology of Computer Vision", McGraw-Hill, pp. 211-280.

Minsky M. (1986): *The Society of Mind*, Simon & Schuster, NY.

Motro A. (1987): *Superviews: Virtual Integration of Multiple Databases*, IEEE Transactions on Software Engineering 13(7), pp. 785-798.

Müller J. /ed./ (1993): *Verteilte künstliche Intelligenz: Methoden und Anwendungen*, BI-Wiss.-Verl., Mannheim, Germany, 393 p.

Newell A. (1982): *The Knowledge Level*, Artificial Intelligence 18(1), pp. 87-127.

Northey P. & Southway N. (1993): *Cycle Time Management: The Fast Track to Time-Based Productivity Improvement*, Productivity Press, Portland, OR.

OMG (1998): *CORBA 2.2 / IIOP Specification, formal / 98-03-01*, Object Management Group Inc. Publication, 95 p.

Orfali R., Harkey D. & Edwards J. (1997): *Instant CORBA*, John Wiley & Sons, 313 p.

Orfali R., Harkey D. & Edwards J. (1999): *Client/Server Survival Guide*, Third ed., John Wiley & Sons, 800 p.

Owen J. (1997): *STEP: An Introduction*, Second ed., Product Data Engineering Series, Information Geometers, Winchester, UK, 224 p.

Özsu M. T. & Valduriez P. (1991) : *Principles of Distributed Database Systems*, Prentice-Hall, Englewood Cliffs, NJ.

Peña-Mora F., Sriram D. & Logcher R. (1995): *Conflict Mitigation System for Collaborative Engineering*, in: "Artificial Intelligence for Engineering Design, Analysis and Manufacturing".

Pohl J. G., La Porta J., Pohl K. J. & Snyder J. (1992): *AEDOT Prototype (1.1): An Implementation of the ICADS Model*, CAD Research Center, Design Institute Rep. CADRU 07-92, California Polytechnic State University. San Luis Obispo, 80 p.

Poyet P. (1993): *XPDI – eXpert Product Data Interchange station*, Technical Report, ILC/93/1314/PP, CSTB, Sophia Antipolis, France, 456 p.

Poyet P., Brisson E., Grivart E., van Horssen J., Behage B., Mooij M., Greening R. & Singh R. (1994a): *Data Abstraction Generalisation and View Conversion Mechanisms*, ATLAS Deliverable D302c, EU/CEC ESPRIT Project 7280, Brussels, Belgium.

Poyet P., Grivart E., Brisson E., Besse G., Irvine M., Greening R. & Böhms M. (1994b): *ATLAS: Implementation of Knowledge Base Extensions*, ATLAS Deliverable D301a, EU/CEC ESPRIT Project 7280, Brussels, Belgium.

Prasad B. E. (1996): *Concurrent Engineering Fundamentals (Integrated Product and Process Organization)*, Prentice-Hall, Englewood Cliffs, NJ.

Quaife C. (1991): *The Hidden Recesses of Time Management*, in: proc. New Delhi Conf. on Project Management for Developing Countries, March 1, New Delhi, India.

Reddy M. P., Prasad B. E., Reddy P. G. & Gupta A. (1994): *A Methodology for Integration of Heterogeneous Databases*, IEEE Transactions on Knowledge and Data Engineering 6(6), pp. 920-933.

Robinson P. (1988): *CIM's Missing Link: Object-Oriented Databases*, Computer Graphics World 10/88, pp. 53-58.

Rose M. T. (1989): *The Open Book: A Practical Perspective on OSI*, Prentice Hall, Englewood Cliffs, NJ, 651 p.

Rumbaugh J. (1996): *A Private Workspace: Why a Shared Repository is Bad for Large Projects*, SIGS Publ., New York, NY, 6 p.

Rumbaugh J., Blaha M., Premerlani W., Eddy F. & Lorensen W. (1991): *Object Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ, 500 p.

Rumbaugh J., Jakobson I. & Booch G. (1998): *The Unified Modeling Language Reference Manual*, Addison-Wesley Object Technology Series, Addison-Wesley, NY, 550 p.

Russel I. S. & Norvig P. (1995): *Artificial Intelligence - A Modern Approach*, Prentice Hall, NJ, 932 p.

Sauce G., Luiten G. T., Watson A. S., Gielingh W. F., van Rijn T. P., Choudhari D. & Hannus M. (1997): *Information Technologies and their Potential Uptake within Large Scale Engineering in Construction*, ELSEWISE ESPRIT Project Report, Oct.’97, CSTB, France, 10 p.

Sauter G. & Käfer W. (1995): *EXPRESS as the Common Data Model in Federated Database Systems*, in: Proc. 5th Int. Conf. of the EXPRESS User Group, EUG’95, Grenoble, France, 20 p.

Scherer R. J. (1995): *The EU Project COMBI - Objectives and Overview*, in: Scherer R. J. (ed.) “Product and Process Modelling in the Building Industry”, Proc. ECPPM’94, Dresden, 5-7 Oct., Balkema, Rotterdam, The Netherlands, pp. 503-510.

Scherer R. J (1997a): *Overview of Requirements and Vision of ToCEE: Annual Report 1996*, EU/CEC ESPRIT Project 20587, Report K1.2 (public), Inst. of Applied Computer Science in Civil Engineering, Dresden Univ. of Technology, Germany, 30 p.

Scherer R. J. (1997b): *Legal Framework for a Virtual Enterprise in the Building Industry*, in: Pawar K. S. (ed.): Proc. 4th Int. Conf. on Concurrent Enterprising, Oct. 97, Nottingham, UK.

Scherer R. J. (1998a): *A Framework for the Concurrent Engineering Environment*, in: Amor R. (ed.) „Product and Process Modelling in the Building Industry“, Proc. ECPPM‘98, Building Research Establishment, Watford, 19-21 Oct., Clowes Group, Beccles, Suffolk, UK, pp. 449-458.

Scherer R. J. (1998b): *AI Methods in Concurrent Engineering*, in: Smith I. (ed.) “Artificial Intelligence in Structural Engineering”, Lecture Notes in AI, Vol. 1454, Springer, pp. 359-384.

Scherer R. J. (2000): *Client-Server System for Concurrent Engineering*, ToCEE Final Report, EU/CEC ESPRIT Project 20587, Inst. of Applied Computer Science in Civil Engineering, Dresden University of Technology, Germany.

Scherer R. J. & Katranuschkov P. (1994): *Integration sollte mehr sein als reiner Datenaustausch*, in: Saal H., Bucak Ö. (eds.) "Neue Entwicklungen im Konstruktiven Ingenieurbau", Festschrift Prof. Mang / Prof. Steinhardt, University Karlsruhe, Germany, pp. 749-762.

Scherer R. J. & Katranuschkov P. (1999): *Knowledge-Based Enhancements to Product Data Server Technology for Concurrent Engineering*, in: Proc. 5th Int. Conf. on Concurrent Enterprising, ICE'99, 16-17 March, The Hague, The Netherlands.

Scherer R. J. & Sparacello H.-M. /eds./ (1996): *COMBI - Final Report*, EU/CEC ESPRIT Project 6609; Research Report 1/96, Inst. of Applied Computer Science in Civil Engineering, Dresden University of Technology, Germany, 60 p.

Schneider S. (1992): *Eine STEP orientierte Objektdatenbank zur Integration von CAD/CAM-Anwendungen*, Ph. D. Thesis, University Karlsruhe, Germany, 133 p.

Schönhoff M., Strässler M. & Dittrich K. R. (1997): *Data Integration in Engineering Environments*, in: Conrad S., Hasselbring W., Heuer A. & Saake G. (eds): Engineering Federated Database Systems EFDBS'97, Proc. Int. CaiSE'97 Workshop, Barcelona, 16-17 June, pp. 45-56.

Schulz R. C. (1996): *Computer Mediated Communications in Architecture, Engineering and Construction*, Res. Report, Dept. Civil Eng., Cal. State Univ., CA.

SDRC (2000): *Metaphase Aerospace and Defense Solution*, White Paper, Structural Dynamics Research Corporation, Milford, OH, 17 p.
(available from: http://www.sdrc.com/pdf/Metaphase_A-D.pdf)

Sedgewick R. (1992): *Algorithmen in C* (german ed.), Addison-Wesley, 742 p.

Serén K.-J., Hannus M., Karstila K., Kihlman M. & Pellosniemi J. (1993): *Object-oriented CAD Tool Implementations for the Construction Industry*, VTT Research Notes 1460, Espoo, Finland, 93 p.

Sherwin D. (1996): *The Estimator*, in: AECNET – The Electronic Resource Network of Architecture, Engineering and Construction, 8/96.

Sheth A. P. & Larson J. A. (1990): *Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases*, ACM Computing Surveys 22(3), pp. 183-236.

Sieberer W. & Keber R. (1997): *The Object World of the REMAP System*, in: Proc. 4th Int. Conf. on Concurrent Enterprising, 8-10 Oct., Nottingham, UK, pp. 195-203.

Smith P. & Reinertsen D. (1991): *Developing Products in Half the Time*, Van Nostrand Reinhold, NY, 296 p.

Sørensen L. S. & Andersen T. (1996): *The Current State of Computing in Building Design and Construction: A Detailed Survey*, in: Kumar B. & Retik A. (eds.) "Information Representation and Delivery in Civil and Structural Engineering Design", Civil-Comp Press, Edinburgh, UK, pp. 171-176.

Spaccapietra S. & Parent C. (1994): *View Integration: A Step Forward in Solving Structural Conflicts*, IEEE Transactions on Knowledge and Data Engineering 6(2), pp. 258-274.

Spaccapietra S., Parent C. & Dupont Y. (1992): *Model Independent Assertions for Integration of Heterogeneous Schemas*, VLDB Journal 1(1), pp. 81-126.

Sriram D. (1991): *Computer Aided Collaborative Product Development*, Research Report IESL-91-05, Dept. of Civil Eng., MIT, MA, 54 p.

Sriram D. & Logcher R. (1996): *The MIT DICE Project*, Computing 26(1), pp. 64-66.

Steele G. L. Jr. (1990): *Common Lisp. The Language*, Second ed., Digital Press, 1031 p.

STEP Tools (1999): *ST-Developer v7 Online Manuals*, STEP Tools Inc.
(available from: http://www.steptools.com/support/stdev_docs/)

Stumpf A. L., Ganeshan R., Chin S. & Liu L. Y. (1996): *Object-Oriented Model for Integrated Construction Product and Process Information*, J. Computing in Civil Engineering 10(3), pp. 204-212.

Svennson K. (1991): *Neutral Building Product Model (The KBS Model)*, Tech. Rep., Swedish National Board of Public Building, Technical Division, Stockholm, 144 p.

Tanenbaum A. S. (1988): *Computer Networks*, 2nd ed., Prentice Hall, Englewood Cliffs, NJ.

Tighe J. (1991): *Benefits of Fast Tracking are a Myth*, Int. J. of Project Management, 9(1), pp. 49-51.

Tsichritzis D. & Klug A. (1978): *The ANSI/X3/SPARC DBMS Framework*, Information Systems, Vol. 3/1978, pp. 173-191.

Thomson A. (1998): *The PIPPIN Data Warehouse Project*, in: Proc. European Conf. "Product Data Technology Days 1998", 25-26 March, Garston, Watford, UK, published by: QMS, Sandhurst, UK, pp. 31-38.

Tolman F. & Poyet P. (1995): *The ATLAS Models*, in: Scherer R. J. (ed.) "Product and Process Modelling in the Building Industry", Proc. ECPPM'94, Dresden, 5-7 Oct., Balkema, Rotterdam, The Netherlands, pp. 473-477.

Turk Ž. (1996): *A Scenario for Building Regulation Processing in a Networked Engineering Environment*, in: Turk Ž. (ed.) „Construction on the Information Highway" Proc. CIB-W78 Workshop, Bled, 10-12 June, CIB Proceedings Publ. 198, University of Slovenia, Ljubljana, pp 503-510.

Turk Ž. (1998a): *On Theoretical Backgrounds of CAD*, in: Smith I.F.C. (ed.) "Structural Engineering Applications of Artificial Intelligence", Lecture Notes in Artificial Intelligence 1454, Springer, Berlin, pp. 490-496.

Turk Ž. (1998b): *Limits of Information Technology in Engineering (Or Why Computers Might Not Replace the Engineers)*, in: Proc. 4th Symposium on Intelligent Systems, Croatian Systems Society, 11th of June, Zagreb, 9 p.

Turk Ž. & Scherer R. J. /ed./ (2000): *Towards a Concurrent Engineering Environment in the Building and Civil Engineering Industries*, ToCEE CD-ROM, EU/CEC ESPRIT Project 20587, Directorate Generale III, Brussels, Belgium.

Turk Ž., Wasserfuhr R. & Katranuschkov P. (2000): *Environment Modelling for Concurrent Engineering*, Int. J. of Computer Integrated Design and Construction /CIDAC/ 2(1), Special Issue on Concurrent Eng. in Construction, SETO, London, UK, pp. 28-36.

Turk Ž., Wasserfuhr R., Katranuschkov P., Scherer R. J., Amor R. & Hannus M. (1997): *Conceptual Modelling of a Concurrent Engineering Environment*, in: Anumba C. J. & Evbuomwan N. F. O. (eds.) „Concurrent Engineering in Construction“, Institution of Civil Engineers, London, UK, pp. 195-205.

Turner J. (1988): *A Systems Approach to the Conceptual Modelling of Buildings*, in: Christiansson P., Karlsson H. (eds.) "Conceptual Modelling of Buildings", CIB Proceedings Publ. 126, Swedish Building Center, Solna, Sweden, pp. 179-187.

Turner J. (1990): *AEC Building Systems Model*, ISO TC184/SC4, N 363, Working Paper, 23 p.

Ullman J. D. (1988): *Principles of Database and Knowledge-Base Systems*, Volume I, Computer Science Press, Rockville, MD, 631 p.

Verhoef M., Liebich T. & Amor R. (1995) *A Multi-Paradigm Mapping Method Survey*, CIB W78 - TG10 Workshop on Modeling of Buildings through their Life-cycle, 21-23 Aug., Stanford University, CA, pp. 233-247.

de Vries B. (1991): *The Minimal Approach*, in: Wagter H. (ed.) Preproceedings CIB W78 Seminar “Computer Integrated Future”, Univ. of Edinburgh, UK.

Ward A., Liker J. K., Cristiano J. L. & Sobek D. K. (1995): *The Second Toyota Paradox: How Delaying Decisions Can Make Better Cars Faster*, in: Sloan Management Review, Spring 1995, MIT, Cambbridge, MA., pp. 43-61.

Waskiewicz F. & Siegel J. (1996): *Establishing a Common Foundation for Product Data Management Software Interoperability*, OMG Publ., in: EDN 9/96.

Wasserfuhr R. & Scherer R.J. (1997): *Information Management in the Concurrent Design Process*, in: Proc. Int. Colloquium IKM’97, 23-25 Feb., Weimar, Germany.

Wasserfuhr R. & Scherer R. J. (1999): *ToCEE – Process and Information Logisitics Services: Documentation of the Server and Tools*, EU/CEC ESPRIT Project 20587, Report E4 (public), Inst. of Applied Computer Science in Civil Engineering, Dresden Univ. of Technology, Germany, 48 p.

Watson A. & Crowley A. (1995): *CIMsteel Integration Standards*, in: Scherer R. J. (ed.) “Product and Process Modelling in the Building Industry”, Proc. ECPPM’94, Dresden, 5-7 Oct., Balkema, Rotterdam, The Netherlands, pp. 491-494.

Weiler K. (1986): *Topological Structures for Geometrical Modelling*, Ph. D. Thesis, Rensselaer Politechnic Institute, Troy, NY, 322 p.

Weise M. (1999): *Konzeption und Validierung eines objekt-orientierten Tragwerkmodells für den Bereich Hochbau auf der Basis des IFC-Projektmodells 2.0*, Diploma Thesis, Inst. of Applied Computer Science in Civil Engineering, Dresden University of Technology, Germany, 132 p.

West M. (1994): *The Data Management Guide. Developing High Quality Data Models*, Vol 1-3, Report No. IC94-033/35, Shell IPC-ICT/47, London, UK.

West M. & Fowler J. (1996): *Developing High Quality Data Models*, Version 2.0, Issue 2.1, EPISTLE Report *(available from: http://www.stepcom.ncl.ac.uk).*

Winner R. I., Pennell J. P, Bertrand H. E. & Slusarezuk M. M. G. (1988): *The Role of Concurrent Engineering in Weapon Systems Acquisition*, Institute for Defense Analysis, IDA Report R-338, Alexandria, VA.

Wittenoom R. (1997): *A Framework for Project Realization*, ISO TC 184/SC4/WG12 Parametrics N082, White Paper, NIST, Gaithersburg, MD.

Wittenoom R. (1998): *Automating Realization of Integrated Project Models*, ISO TC184/ SC4/WG12/Parametrics N192, 20 p.

Wix J. (1996): *Purpose of a Core Model*, Tech. Report, Research Project „Computerised Exchange of Information in Construction", Dept. of the Environment, UK, 12 p. *(available from: http://helios.bre.co.uk/ccit/info/bccm/purpose.htm)*

Womack J. P., Jones D. T. & Roos D. (1991): *The Machine That Changed the World: The Story of Lean Production*, Harper Perennial, NY.

Woods W. A. (1991): *Understanding Subsumption and Taxonomy: a Framework for Progress*, in: Sowa J. (ed.) „Semantic Networks: Explorations in the Representation of Knowledge", Morgan Kaufmann, San Mateo, CA.

Yang Y. (1995): *STEP Application Protocol Implementation*, Tech. Report, CIC Group, Building and Fire Research Laboratory, NIST, Gaithersburg, MD, 22 p.

Zarli A. (1995): *XP-Rule: A Language for the Representation of Kno*wledge, Tech. Report, CSTB, Sophia Antipolis, France.

Ziemke C. & Spann M. (1993): *Concurrent Engineering's Roots in the World War II Era*, in: Parsaei H. R. & Sullivan W. G. (ed.): "Concurrent Engineering: Contemporary Issues and Modern Design Tools", Chapman & Hall, London, UK, pp. 24-41.

Zucker J. & Demaid A. (1992): *Modelling Heterogeneous Engineering Knowledge as Transactions Between Delegating Objects*, in: Gero J. S. (ed.) "Artificial Intelligence in Design '92", Kluwer Academic Publ., Dordrecht, The Netherlands, pp. 141-159.

Glossary

The basic abbreviations used in this thesis are as follows:

Term	*Meaning*	*Area*
ACL	Access Control List	Information Technology
AEC	Architecture, Engineering, Construction	Building Construction
AI	Artificial Intelligence	Computer Science
AIC	Application Interpreted Constructs	ISO STEP
AIM	Application Interpreted Model	ISO STEP
AP	Application Protocol	ISO STEP
API	Application Program Interface	Information Technology
ARM	Application Reference Model	ISO STEP
BLOB	Binary Large Object	Information Technology (IT)
CAD	Computer Aided Design	Information Technology in all industry areas
CAE	Computer-Aided Engineering	Building IT
CAx	General term for CAD, CAM, CAE etc.	Information Technology in all industry areas
CALS	Continuous Acquisition and Life-Cycle Support	Information Technology in all industry areas
CE	Concurrent Engineering	Abbrev. used in this thesis
CEE	Concurrent Engineering Environment	Abbrev. used in this thesis
CGI	Common Gateway Interface	Information Technology
CIC	Computer Integrated Construction (corresponds to CIM in mechanical eng.)	Building IT
CLOS	Common LISP Object System	Information Technology
CORBA	Common Object Request Broker Architecture	Information Technology
CSML	Context-Independent Schema Mapping Language	Acronym from this thesis
DBMS	Database Management System	Information Technology
EBNF	Extended Backus-Naur format	Information Technology
EDI	Electronic Data Interchange	Information Technology in the business domain
EDIFACT	International standard for EDI messages	Information Technology in the business domain
EDM	Electronic Document Management	Information Technology
ER	Widely used abbreviation for the Entity-Relationship modelling paradigm	Information Technology
FM	Facilities Management	Building Construction
FTP	File Transfer Protocol	Information Technology

Term	*Meaning*	*Area*
GUI	Graphical User Interface	Information Technology
HTML	Hypertext Mark-Up Language	Information Technology
HTTP	Hypertext Transfer Protocol	Information Technology
HVAC	Heating, Ventilation and Air Conditioning	Building Construction
IAI	International Alliance for Interoperability	Building IT
IDL	Interface Definition Language for CORBA	Information Technology
IFC	Industry Foundation Classes (the modelling framework developed by the IAI as a de facto industry standard for the building industry)	Building IT
IR	Integrated Resources	ISO STEP
NIAM	Nijssen's Information Analysis Method	Information Technology
ODBC	Open Data Base Connectivity	Information Technology
OMT	Object Modelling Technique	Information Technology
OSI	Open Systems Interconnection	Information Technology
PDM	Product Data Management	Information Technology in all industry areas
PDT	Product Data Technology	Information Technology in all industry areas
PROMISE	Product Data Management Information Server	Acronym from this thesis
RDBMS	Relational Database Management System	Information Technology
RFC	Request For Change	Building Construction
RMI	Remote Method Invocation (as introduced in the Java language)	Information Technology
SDAI	Standard Data Access Interface	ISO STEP
SGML	Standard Generalised Mark-Up Language	Information Technology
SPF	Unofficial, but widely used abbreviation for the STEP physical file format	ISO STEP
SQL	Structured Query Language, a standardised interface specification for database management systems	Information Technology
STEP	Standard for the Exchange of Product Data	Information Technology in all industry areas
TCP/IP	Transmission Control Protocol / Internet Protocol (the basic protocol family of the Internet)	Information Technology
UML	Unified Modelling Language	Information Technology
VR	Virtual Reality	Information Technology
VRML	Virtual Reality Modelling Language	Information Technology
WfM	Workflow Management	Information Technology
WfMC	The Workflow Management Coalition	Information Technology
WWW	The World Wide Web	Information Technology in all areas
XML	Extensible Mark-Up Language	Information Technology
XML DTD	XML Document Type Definition	Information Technology

Indices

Index of Key Terms

Quick Reference Index

The following lists are provided for easier reference to figures, tables, EXPRESS-G and other diagrams, specifications, examples and quotations in the text.

List of figures

List of tables

List of diagrams in formal graphical notation

List of specifications

Index of authors

This index presents the names of the authors of the referenced literature sources as cited in the text (first or first and second author).