Construction Informatics

issues in engineering, computer science, and ontology

Asger Eir

Kongens Lyngby 2004
IMM-PHD-2004-131

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

IMM-PHD: ISSN 0909-3192

To Rikke — for love,
support, and encouragement

Preface

This Ph.D.–thesis studies issues in the area of *construction informatics*. Construction informatics is the theoretical study of formal and conceptual aspects in the domain of civil engineering and design.

The thesis is a collection of papers which each treat a specific subject within domain analysis and conceptual modelling of civil engineering and design.

Due to the interdisciplinary content, the first half of the study has been carried out at Department of Civil Engineering (BYG•DTU), The Technical University of Denmark; whereas the second half has been carried out at Informatics and Mathematical Modelling, The Technical University of Denmark. Supervisors have been Prof. Dines Bjørner (IMM) and Per Galle (BYG•DTU).

The idea was to initiate the study at a place where engineering issues are discussed on a daily basis, and where the practical and theoretical knowledge of the domain is present.

With origin in civil engineering and design issues, the study was directed towards computer science oriented theories in an attempt to introduce such theories in modelling and clarification of the domain. This strategy turned out to be a strength for the study and this thesis. However, it also discovered some problems in carrying out such a truly interdisciplinary Ph.D.–study. Per Galle's and Dines Bjørner's common background in computer science has been essential for the success of this study.

The original title of the Ph.D. project was "*Design and application of a civil engineering ontology* However, it became clear that there were going to be two

main streams in the thesis, and that an actual monograph was not an appropriate format for the thesis.

The main streams are both rooted in civil engineering ontology, and they are bound together by the overall issue of how civil engineering concepts relate.

The issues of the thesis are treated from three angles: from computer science, from civil engineering and design theory, and from philosophy. It is characteristic for the thesis that these angles are all present in analysis and argumentation. The philosophical aspect is a natural ingredient as construction informatics primarily concerns the fundamental conceptual structures, and how models of these relate to engineering and design practice and reality.

The aspect of design has been given high priority because this subject concerns the relation between representation and artefacts — a subject which is also essential in computer science, and which is deeply rooted in philosophy.

Asger Eir

Lyngby, February 2004

Forord

Denne Ph.d.–afhandling studerer emner inden for *byggeinformatik*. Byggeinformatik er det teoretiske studie af formelle og begrebsmæssige aspekter i genstandsområdet byggeri og design.

Afhandlingen er en samling af artikler, som hver behandler et afgrænset emne inden for domæneanalyse og begrebsmodellering af byggeri og design.

Grundet studiets tværfaglige indhold, er første halvdel udført på BYG•DTU og anden halvdel på Informatik og Matematisk Modellering (IMM) med Prof. Dines Bjørner (IMM) og Per Galle (BYG•DTU) som vejledere.

Ideen var at starte studiet der, hvor de ingeniørmæssige problemstillinger blev dagligt diskuteret, og hvor man havde den praktiske erfaring og teoretiske viden om genstandsområdet. Fra de bygge– og designteoretiske studier drejede studiet sig til de mere formelt datalogiske emner i et forsøg på at indføre disse i modellering og afklaring af genstandsområdet. Dette forløb har vist sig at blive en styrke for studiet og denne afhandling, men det har også afsløret problemstillinger i at gennemføre sådanne virkelig tværfaglige Ph.d.–studier. Per Galles og Dines Bjørners fælles baggrund inden for datalogi har været essentielt for studiets succes.

Den oprindelige titel på Ph.d.–projektet var "*Design og anvendelse af en byggeontologi*. Det viste sig imidlertid hurtigt, at der tegnede sig individuelle hovedlinier og en egentlig monografi ville derfor ikke være en naturlig form for afhandlingen. Hovedlinierne har dog alle rod i det byggeontologiske og er bundet sammen af den overordnede problemstilling om, hvorledes byggebegreber relaterer.

Emnerne i afhandlingen behandles med udgangspunkt i tre vinkler: en datalogisk, en bygge–design teoretisk, og en filosofisk. Det er kendetegnende for afhandlingen, at disse tre vinkler er til stede i analyse og argumentation. Det filosofiske aspekt indgår som et naturligt element da byggeinformatik først og fremmest omhandler basale begrebsmæssige strukturer, og hvorledes modeller af disse relaterer til den ingeniør– og designmæssige virkelighed. Designaspektet har ligeledes fået stor vægt, da dette emne særlig handler om relationen mellem repræsentationer og artefakter — et emne, som både er centralt i datalogien og dybt rodfæstet i filosofien.

Asger Eir

Lyngby, Februar 2004

Acknowledgement

My sincere thanks go to my supervisors Prof. Dines Bjørner and Per Galle for their tremendous inspiration and support.

Dines inspired me — quite early in the process — to think of denotational semantics and language orientation; two approaches which have been fundamental for the study and the results. His insightful research in the whole methodology of computing science — and especially the focus on domain engineering — has been an essential foundation for all my ideas. His influential ideas and perspectives have encouraged me to apply informatic thinking in the broad. For this — and much more — I am truly grateful.

Per encouraged me from the start to study basic philosophy of logic and language. With deep insight and an ability to apply this insight in various areas like design, he has taught me a whole new dimension — a dimension which has been essential for the thesis and for my personal development.

I want to direct special thanks to Prof. Anders Ekholm, Rob Howard, and Flemming Vestergaard for countless discussions on construction IT, classification, and design. Anders Ekholm provided for me to stay two month at Lund University working in his group. His design–philosophical insight has been a great inspiration to me, and our collaboration — leading to the paper in Chapter 4— has been an important break through for me.

I will also like to thank Nikolaj Oldager for our discussions on ontology, and Prof. John Mylopoulos, University of Toronto, for making a four month stay possible. I also thank John for our many discussions and for his critique which

provided me with new angles on my work.

I am also grateful to Anjan Chakravartty and Gurpreet Rattan, Department of Philosophy, University of Toronto, for explaining to me many issues in metaphysics and philosophy of language. Also, I thank Anjan for our discussions on properties, causation, and our common interest in Shoemaker's ontology.

Special warm thanks are due to Davide Bolchini for countless discussions on modelling, and for being truly supportive at hard times.

Furthermore, I thank Tamer El–Diraby, Department of Civil Engineering, University of Toronto, for our discussions on domain modelling and ontologies in civil engineering. His enthusiasm with the interdisciplinary field of construction informatics has been a great encouragement. While visiting Toronto, Tamer arranged that I could give some presentations for his group. The feedback I got, helped me shape my ideas and strengthen my argumentation.

I have been so lucky to be surrounded by persons possessing high expertise on various fields. Thus, I thank Jørgen Steensgaard-Madsen and Hans Bruun for answering my many questions on programming languages and semantics, Anne Haxthausen and Morten P. Lindegaard for assistance on RSL, Michael R. Hansen for helping me with SML issues, Jørgen Fischer Nielsson for answering various questions and for suggesting important readings on philosophy and ontology. Thanks are also due to Tom Østerby for his constant interest in ontology.

Furthermore, I thank Prof. Steve Easterbrook, University of Toronto, for his insightful critique on my crazy ideas on design languages and semantic parameterised interpretation. Likewise, I thank Michael Jackson, and Rick Hehner, University of Toronto, for their constructive and insightful responses on my work and ideas.

Thanks are also due to Leif Sjøgren, Sund & Belt A/S, Niels–Jørgen Gimsing, Jan Lambeck, Thomas Bolander, Sidney Shoemaker, Victoria Weafer, Ole Eir, Charlotte Eir, Christian Kragholm, and Rikke Søholm Bønnelykke Eir.

Finally, I thank the foundations of Nordisk Forskningsakademi (NorFA), Frants Alling, and Reinholdt W. Jorck, who provided financial support to my stays at Lund University and University of Toronto.

Contents

Part I

Opening

Chapter 1

Introduction

This thesis studies issues in the interdisciplinary area between civil engineering and computer science. The formal aspects of this study, we call "*Construction Informatics*". As the name indicates, we are concerned with the domain of construction (civil engineering and design) and approach it in an informatic way. Informatics is the convergence of computer science, mathematics (including mathematical modelling), and applications. Thereby, construction informatics is the theoretical study of the mathematical abstractions which can be taken to model construction domain concepts. It is an interdisciplinary field which roots in the problems of civil engineering and design, as well as in the problems of representation and computation.

Today, information technology (IT) is commonly used in several areas of construction and lately it has become a fast growing research topic as well. The last ten years of research has drawn on results from computer science theory and practice. Such efforts include defining classification systems or *ontologies*, introducing databases for collaborative design, defining core product models for tool integration, constructing and applying project web services, and introducing portable communication facilities. We call this line of work "*IT in construction*" in order to emphasize its focus on technology. As opposed to this, "*construction informatics*" is the theoretical study of foundations and is based on mathematics, abstraction, and philosophy.

Industry and research have emphasized a number of major challenges to future information technology in the construction sector. One can be formulated as the motto of "*getting information, wherever you are — whenever you want*". This challenge concerns communication, integration, and standardisation. Another challenge is how to better support the work of practitioners. This is an issue which is rooted in the nature — the intrinsics — of the domain of civil engineering and design.

> Construction informatics is the formal study which investigates the domain of construction and its conceptual foundations by introducing computer science concepts of theoretical kinds. In this thesis, we shall do so by focussing on the very basic structures — the intrinsics — of certain facets of civil engineering and design. The thesis is a series of papers which treat specific construction informatic issues.

What governs our approach is partly the principle of specifying domain concepts as formal models, and partly the philosophical considerations on which domain clarifications are founded. Thereby, we believe to touch issues which are essentially important when facing future challenges in civil engineering and design — practical as well as research oriented.

Construction of buildings is a traditional and conservative industry which differs from other production businesses on the amount of information, the complexity of organisations, and the uniqueness of the products. Construction projects involve a large number of stakeholders who often use different tools, conventions for representation, rules & regulations, and means for communication. Furthermore, the amount of information in the construction industry is enormous and many–sorted. It includes construction specifications, drawings, contracts, schedules, budgets, information for facilities management, etc. The meaning and significance of such information is not uniquely defined and often depend on the rôle of the given stakeholder.

With todays distributed trades, the area of construction is going from being *product–oriented* to be *service–oriented*. The reason is that focus has moved from the building as a product, to the information and services in the building project. This means that notions like *classification*, *design*, *project management*, etc., need to be founded on much more fundamental conceptual structures than previously. Such structures are rooted in the *intrinsics* of the domain; not in syntactical conventions or currently convenient practice.

From an engineering perspective, the domain of civil engineering and design is interesting because it comprises notions like language, description, representa-

tion, and communication; and these notions relations to physical or possible artefacts.

From a perspective of computer science and informatics, the domain is interesting because it comprises huge amounts of information in documents that seem to be related semantically. Some documents describe the artefact — the building — to be built. From a semantic perspective, the descriptions in such documents are not simple, as they refer to things or phenomena which may or may not exist. In general, dealing with information in civil engineering may lead to considerations of ontological and philosophical kinds. It does so by including notions like *properties*, *representation*, *mereology*, and the meaning of language constructs.

Thus, an investigation of the domain of civil engineering contributes to: (i) a conceptual clarification of the domain in general, (ii) an understanding of the domain as a foundation for developing information systems, (iii) an understanding of and experience with the computer science methodology applied in the process, and (iv) an awareness of the significance of formal models.

The present thesis aims at reaching a clarification on certain facets of the domain of civil engineering and design. This clarification process is constantly flavoured with: (i) domain intrinsics and problem issues, (ii) computer science concepts and principles, and (iii) philosophical and ontological considerations. Thus, our analysis and treatment is constituted by three angles: *Civil engineering and design*, *Computer science* and *Philosophy*:

- A domain clarification of civil engineering and design, is necessarily rooted in observations and conceptions of what is going on, the problems occurring, and the approaches taken to accommodate them. It is a study deeply rooted in the notion of representation and the relation between representation and the represented. In this context, notions like physical entities and mental ideas of buildings are important. So are the process of designing, the complexity in managing construction information, etc.
- The computer science angle is rooted in computer science and mathematics which in this thesis means that we use well–known concepts like formal models, orderings, lattices, formal semantics, etc. It is essential to the study that we strive towards full formalisation of the domain concepts considered. This, we take as a criterion for the conceptualisation of the domain to be useful as foundation for advanced software systems.
- All domain considerations are based on conceptions of the world. Philosophy is, however, not a solution schema which makes things "run". It is an exploration process which may introduce more questions than answers.

Therefore, philosophy is often considered too theoretical for actual applications in or solutions to theoretical issues. Still, philosophy is the study of foundations and is thereby essential in everything we do. In this thesis, we have put quite an emphasis on philosophical considerations. We are interested in arguing *why* things are as they are and thus we shall not be content with explanations which refer to conventions. The pragmatic question of *why* is what drives the philosophical considerations and thus it should deliberately drive the domain clarification process as well.

1.1 Audience and prerequisites

The work presented in this thesis is aimed at people who work in research or with industrial treatment of computer science application domains. Primarily, it is aimed at people working in the interdisciplinary field of construction and informatics. We hope that the thesis may be appreciated as an approach which formally investigates this field. We see it as a contribution, both to research of civil engineering ontologies and of information systems for various purposes within the field. Although we do not present an actual ontology, we believe that the methodology and formal theoretical foundation presented may be beneficial to research and development in the area of civil engineering ontologies, classification systems, standardisation, tool integration, and information systems in general. Studies of the foundation for information systems includes the study of computer aided design. In this area, we believe that the thesis contributes with important considerations, clarifications, and solutions.

In the area of computer science, the thesis can be considered an example of domain engineering. Thereby, it is a collection of studies in a large series which collectively aim at reaching clarifications on and experience with the methodology for domain acquisition. Also, the thesis may be relevant to computer scientists who are interested in the rôle played by philosophy in this context.

The philosophical aspects may be of interest and relevance to people working with similar ontological problems in civil engineering, in design, or in other domains. However, the way we use philosophy is quite specialised towards civil engineering and design. The contributions on the philosophical front should therefore be seen as contributions with respect to the given domain. They may not be actual contributions to philosophy themselves, and thus may not interest philosophers who are experts on the areas being touched. What *may* be of interest is, however, the way we utilize philosophy as an actual beneficial foundation study for solving technical problems.

In the thesis we take a so–called *language–oriented* approach to modelling. This approach is based on the distinction among the semiotic notions of *pragmatics*, *semantics*, and *syntax*. It is strongly recommended to have an understanding of this distinction for reading the thesis.

Throughout the thesis, we make use of formal specifications, primarily in *The RAISE Specification Language* (RSL [134, 135]). Such specifications are precise mathematical formulations of the ideas being presented. The specifications refer to notions like sets, maps, functions, types, etc. A basic understanding of such notions may be needed in order to fully understand the contributions. However, we have made an effort to make our presentations such that the formal specifi-

cations are supplementary. Still, it is advisable to read the presentations with some background knowledge of mathematical abstraction, types and functions, and of logic. A deeper understanding of formulae and proofs requires knowledge of specification languages like RSL, VDM, or similar.

In the paper presented in Chapter 7, we use direct denotational style for specifying the semantics of some languages. Thus, it is advisable to have a good understanding of this notation, as well as of denotational semantics in general (we refer to [161, 127, 148]).

Often, we shall refer to notions like *objects*, *properties*, *concepts*, and *relations*. We consider it essential for the understanding, to have a basic understanding of these notions.

The papers in Chapter 8 and Chapter 9 are of philosophical kind. Hence, they differ from the other chapters with respect to style and background knowledge required. The two papers can be read with some basic knowledge of syntax and semantics. It is advisable to have some experience with reading philosophy, though. For a real benefit of the papers, we recommend that these are read with some background knowledge of the classical problem issues concerning objects, properties, descriptions, meaning, and language.

1.2 Hypothesis (scientific statement)

We shall make a distinction between the term "*hypothesis*" and the term "*thesis*". The former we take to name the formulation of the scientific statement with which we shall be concerned. The latter we take to name this work which describes the approaches, solutions, and results of investigating the hypothesis. The hypothesis is defines such that it with most certainty can be refuted. The overall contribution of the thesis is then the results of exploring to what extent the hypothesis is valid.

We base the thesis on two convictions or *dogmas*; one from computer science and one from cognitive science in civil engineering and design:

A dogma in computer science

Domain engineering is the theoretical study which — with origin in observation and considerations of a domain — establishes models of that domain. Domain engineering is a prerequisite to requirements and design of software systems. Making models of a specific application domain, provides the basis for a better understanding of that domain and thus for making software systems rooted in the nature of the domain.

A dogma in civil engineering and design

The domain of civil engineering and design is a domain of communication processes going from needs and ideas for solutions, via requirements and design, to construction, maintenance, and demolition.

The overall hypothesis of our thesis is now the following:

Hypothesis

Civil engineering concepts can — as formal computable models — be bound together by relations which explicitly specify how information is created, used, and how it evolves through stages of civil engineering projects.

From the hypothesis, we derive our overall motivation:

Motivation

Establishing such relations between civil engineering concepts adds conceptual transparency and clarity to domain models, such that these models make solid foundations for civil engineering information systems.

In the thesis, we shall exercise the hypothesis from four different angles:

① **Relating concepts of different incomparable kinds.**

Our focus will here be how the notion of Galois connections can be used to relate two different concepts. We approach from this angle in Chapter 3.

② **Relating representations of increasing cognitive significance.**

Our focus will here be design processes and design representations. We approach from this angle in Chapter 4, Chapter 5, and Chapter 6.

③ **Conceptual design models versus perspectives (*views*).**

Our focus will here be design tools and their software architectures. We approach from this angle in Chapter 4 and Chapter 7.

④ **On the relation between descriptions and artefacts.**

Our focus will here be on the relation between descriptions and part–whole relations, and on the notion of properties and meaning in context of important design related problems. We approach from this angle in Chapter 8 and Chapter 9.

These angles represent the subjects into which the papers of this thesis are categorised. In Chapter 11, we shall compare our overall results for each of these angles with the hypothesis. The papers constituting the thesis, individually define specialisations of the hypothesis and motivation. Thereby, they can be read as separate research contributions as well.

1.3 Contributions (English)

The primary research contributions of the work in this thesis are:

1. A principle for relating civil engineering domain concepts. This contribution is a result of exercising the hypothesis from angle ①.
2. A formal foundation for incremental design and the introduction of the concept: *design lattices*. This contribution is a result of exercising the hypothesis from angle ②.
3. A principle of *semantic parameterised interpretation* as a new software architecture for conceptual design systems. This contribution is a result of exercising the hypothesis from angle ③.
4. A suggestion of a metaphysical notion: *object aspects*. This contribution is a result of exercising the hypothesis from angle ④.
5. A clarification on the philosophical foundations for design. This contribution is another result of exercising the hypothesis from angle ④.

The items 1.–3. consider the practical problems of handling civil engineering information and the theoretical problems of design. Theories and concepts from computer science are applied in solving these problems.

The items 4.–5. consider the practical and philosophical problems in context of civil engineering and design. Philosophical analysis and theory are here applied in approaching clarifications on the subject matter.

In the following, we describe the contributions as brief introductions to the individual papers of the thesis. For each contribution described, we state the relevance to industry and to other research.

1.3.1 Relating civil engineering domain concepts

The two civil engineering concepts *cost frame* and *project plan* — which can be modelled and understood individually — are related by the mathematical notion of *Galois connections*. Given mathematical models of the two concepts, it is possible to determine which project plans are executable within a given cost frame, and which cost frames apply to a given project plan. Specifying how two such civil engineering concepts relate, implies specifying how knowledge

is built through stages of a building project. In trying to specify the relation between two civil engineering concepts, we may discover that these cannot be related directly. There may be interrelating concepts which bind them together. The principle of relating concepts by means of Galois connections thus includes investigation of what notions tie the concepts together. This is done for the two concepts: *cost frame* and *project plan*.

> *Relevance to industry and research*: The principle can be used as a method with which we can model the relations between various civil engineering concepts. Thereby, information in different project stages and of different kinds, can be linked. That is, we can link information about needs and ideas, requirements and design, process planning and execution. Often documentation is written on the basis of various sorts of knowledge. The principle described tries to make such knowledge explicit and precise by means of formal — i.e. mathematical — specifications. In order to write a project plan we need to know the cost frame, and in order to find a suitable location of a building we need to know the approximate size of the building, etc. Here, computer aided knowledge management in building may benefit from modelling the relations between civil engineering concepts, explicitly. Thereby, we have a method for testing various decisions taken. The main idea is thus: The knowledge necessary for documenting a civil engineering project should be made explicit and precise such that this knowledge can contribute to the management and control of the given project.

1.3.2 Design lattices

The design process can be considered as an exploration and configuration process which can be captured as spanning a lattice structure. This means that it is possible to define an ordering relation between design representation on various stages of development. In a sense, a design process can be considered as a collection of choices and design compositions. E.g. in the design of a load–bearing beam, we may choose among different dimensions and materials, and combination of the properties which are necessary for the beam to possess a certain strength. We have developed a mathematical model of design representations and specified an ordering relation between such representations. Thereby, we have the ability to express that one design representation is more precise than another. Being more precise here means that it contributes with more knowledge of the artefact in mind; i.e. it is cognitively more sufficient. The idea is called *design lattices*. Design lattices are adequate for supporting what

is known as incremental design. By incrementality, we understand that objects, properties of objects, and relations between objects, alternately can be added to a design representation. The notion of design lattices, and its application as a foundation for conceptual design tools, is presented in the two papers Chapter 5 and Chapter 6.

Relevance to industry and research: By recording design processes and representing these as design lattices, we are able to browse between tentative designs and previous designs stages. In a sense, we also have the opportunity to structure the design process better, although this is not the primary aim. In today's design tools, the design process is considered a sequence of object instantiations and removals. The structured designer may want to be aware of the design changes being made, as well as know when one design is more specialised than another, whether two designs are in conflict, can be combined, etc. The notion of design lattices facilitate such functionality without obstructing the creative process of designing.

1.3.3 Semantic parameterised interpretation

The principle of *semantic parameterised interpretation* is introduced as a new software architecture for tools aimed for conceptual building design.

The idea is to make it possible to specify the meaning of terms representing properties which are referred to by names in design models. Such a specification is here called a *semantics*. Design models are now expressed in a special modelling language. If new names for properties are needed in order to express the design idea in mind, the meaning of these names are to be specified in the semantics. A semantics is written in a specially designed specification language. A design model can be interpreted according to the semantics specified. The result is one of many presentations of — views on — the model. Examples of such views could be representations of visualisation commands for displaying the object from various angles, or expressions used in stress analysis of the artefact being modelled.

Relevance to industry and research: Within research of *incremental design*, the design process is considered as a process in which objects, properties of objects, and relations between objects are added incrementally to a design representation. In the paper Chapter 4, we argue that tools for conceptual modelling of buildings must support such incrementality, and that it should be possible to introduce

names for properties when these names are needed in order to express the design idea in mind. Such functionalities are not supported by today's commercial design tools. If we wish the sort of dynamics without having to restructure the type system repeatedly, we need to specify the meaning of the names separately from the design program (including its type structures). The principle of *semantic parameterised interpretation* investigates the possibilities for doing so.

In addition to the theoretical study, a prototype tool has been developed. This tool demonstrates the principle of semantic parameterised interpretation. The tool has been programmed in Moscow ML.

1.3.4 Object aspects

References to physically or potentially existing objects like buildings can be found many forms of building documentation. Words and phrases, which are taken to refer to such objects, do so in two ways. One way is by referring to concepts of which the object in question is considered to fall under. The terms referring to the concepts, plays the rôle of characterising that object. The other way is by referring to another object to which the object in question stands a certain relation. An important one of such relations is the relation between part and wholes. The formal–philosophical theory of part–whole relations is known as *mereology*. In order to solve a number of reference problems, when considering objects which do not have physical presence (like in designing), the notion of *object aspects* is introduced. The existence of the notion — being a special kind of the mereological notion of *parts* — is defended against standard criticisms directed towards mereology.

Relevance to industry and research: The work is a contribution to the understanding of the possibilities and limitations related to documentation and other descriptions of physical things.

1.3.5 Metaphysical theories as foundation for design

A theory of design necessarily needs clarification on three issues: (i) what it means to describe, (ii) how we can describe objects which have no physical presence, and (iii) on what basis we can predict the behaviour of artefacts being designed. We show how a collection of philosophical theories concerned with

language, meaning, and properties, contribute to an understanding of design. In essence, we dig into the aspects of semantics in relation to descriptions of objects.

> *Relevance to industry and research*: In the development and application of design tools and methods, we may often ask the question of whether the knowledge being expressed is merely a collection of commonly agreed symbols. The issue becomes important in context of interoperability between applications as a common language or model is needed. The question is now on what ontological basis such a language or model is to be established. New philosophical theories in metaphysics connect the notion of properties tightly with the notion of causation. From the knowledge of a set of properties we can usually say something about the behaviour of the object possessing these properties; e.g. that pylons for a bridge can take a certain tension. We may apply similar kinds of judgements over objects which are being designed and thus ascribed a set of properties. Imagine that such knowledge was built into computerized design tools; including a large set of natural laws. Thereby, we are able, not only to verify designs against their requirements, but also to simulate the artefact's behaviour when put in certain situations. Special programs can do something like this. We suggest that it all is merged in a conceptual design tool.

Common for all contributions is the problem of how information relates and how it evolves and is used as foundation for documentation through all the stages of a civil engineering project. Thereby, the work is a study in ontology and how to apply ontology as foundation for new technology.

1.4 Bidrag (Danish)

De vigtigste forskningsmæssige bidrag i denne afhandling omfatter emnerne:

1. Et princip for relatering af byggebegreber. Dette bidrag er et resultat af at udforske hypotesen from vinkel ①.
2. Et formelt fundament for inkrementalitet i designprocessen samt introduktion af begrebet: *designgitre*. Dette bidrag er et resultat af at udforske hypotesen fra vinkel ②.
3. Princippet *semantisk parametriseret fortolkning* som en ny software–arkitektur for begrebsmæssige designværktøjer. Dette bidrag er et resultat af at udforske hypotesen fra vinkel ③.
4. En introduktion af det metafysiske begreb: *objektaspekt*. Dette bidrag er et resultat af at udforske hypotesen fra vinkel ④.
5. Afklaringer af en række filosofiske fundamenter for design. Dette bidrag er ligeledes et resultat af at udforske hypotesen fra vinkel ④.

Punkterne 1.–3. tager udgangspunkt i praktiske informations–håndteringsmæssige og designteoretiske problemstillinger, og anvender datalogiske teorier til løsning af disse.

Punkterne 4.–5. tager udgangspunkt i praktiske såvel som videnskabsteoretiske / filosofiske og såkaldt ontologiske problemstillinger, og anvender filosofien til analyse af praktiske problemstillinger.

I det følgende beskriver disse bidrag, idet begrundelser mht. industri– og forskningsmæssig relevans er givet i kursiv efter hver beskrivelse.

1.4.1 Relatering af byggebegreber

De to byggebegreber *udgiftsramme* og *projektplan* — der kan forstås og modelleres hver for sig — spiller sammen vha. det matematiske begreb *Galois connection*. Givet matematiske modeller af de to begreber, er det muligt at afgøre, hvilke projektplaner, som kan udføres inden for en given udgiftsramme, og hvilke udgiftsrammer, som kan anvendes på en given projektplan. At specificere, hvorledes byggebegreber relaterer, vil desuden sige at specificere, hvordan viden opbygges gennem stadierne i et byggeprojekt. I specificering af relationen mellem to byggebegreber vil man ofte opdage, at disse ikke altid kan

relateres direkte. Der kan være begreber, som binder dem sammen. Princippet består således i dels at modellere de involverede begreber, dels at afgøre, hvilke størrelser der binder begreberne sammen. Dette er gjort for begreberne *udgiftsramme* og *projektplan*.

> *Relevans for industri og forskning*: Med princippet i hånden, kan man begynde at modellere relationerne mellem andre byggebegreber således, at information på de forskellige stadier kædes sammen; dvs. fra behov og idé via krav og design til procesplanlægning og udførelse. Når byggedokumentation nedskrives, gøres det oftest på baggrund af viden på en lang række områder. Det beskrevne princip søger at gøre denne viden tydelig og præcis vha. formel — dvs. matematisk — specifikation. For at kunne nedskrive en projektplan er det nødvendigt at kende udgiftsrammen, og for at finde en passende lokalisering, er det nødvendigt at kende til eventuelle bindinger til omgivelserne, byplanlægning, ledige byggegrunde, osv. Datamatunderstøttelse af videnhåndtering i byggeriet kan derfor have gavn af at udtrykke relationerne mellem forskellige byggebegreber. Herved fås desuden værktøjer til at teste de forskellige beslutninger, som tages. Baggrunden for princippet er altså: Den viden, som er nødvendig for at dokumentere et byggeprojekt, bør gøres tydelig og præcis, så denne viden kan bidrage til bedre at styre det pågældende projekt.

1.4.2 Designgitre

Designprocessen kan opfattes som en søge– og konfigurationsproces, der kan repræsenteres som en gitterstruktur (mat. eng: *Lattice*). Det vil sige, at det er muligt at definere en ordningsrelation mellem forskellige designstadier. En designproces består således både af valg mellem forskellige alternativer og af sammensætning af forskellige midlertidige designs. Eksempelvis kan designprocessen for en bærende bjælke omfatte valg mellem forskellige alternative dimensioner og materialer, og sammensætning af de egenskaber, der skal til for, at bjælken får den bæreevne, som er tiltænkt. Der er blevet specificeret en matematisk model for designrepræsentationer samt en ordningsrelation mellem sådanne. Herved kan vi udtrykke, at éen designrepræsentation er mere præcis end en anden; dvs. den bidrager med mere viden om det pågældende artefakt, end den forrige. Princippet kaldes *designgitre* (eng. *Design Lattices*). Designgitre er født til at understøtte det som kaldes *design–inkrementalitet*, hvilket vil sige, at objekter, egenskaber for objekter, og relationer mellem objekter, successivt kan tilføjes en designrepræsentation. Princippet om designgitre præsenteres og specificeres på

to forskellige måder i hver sin artikel (Chapter 5 and Chapter 6).

Relevans for industri og forskning: Ved at "fange" designprocesser og designtrin og relatere dem i et gitter, bliver det muligt at "bladre" (eng: *browse*) mellem forskellige designtrin. Yderligere giver det mulighed for bedre at strukturere designprocessen, om end dette ikke er det primære formål. I nutidens designværktøjer, betragtes designprocessen som en sekvens af tilføjelser og sletninger af objekter. Den strukturerede designer bør gøre sig sine designtrin bevidst; dvs. vide hvornår et design er mere specialiseret end et andet, om to forskellige designløsninger er i konflikt med hinanden, kan kombineres, osv. Designgitre tilbyder muligheden for at indbygge dette i designværktøjer uden at begrænse designerens kreative proces.

1.4.3 Semantisk parametriseret fortolkning

Princippet *semantisk parametriseret fortolkning* introduceres som en ny software–arkitektur for begrebsmæssige værktøjer for eksempelvis design og projektering af bygninger. Idéen er at gøre det muligt at specificere betydningen af egenskaber, der refereres til med navne i designmodeller. En sådan specifikation kaldes her en *semantik*. Designmodeller udtrykkes nu i et særlig konstrueret modelleringssprog. Ved indførelse af nye navne for egenskaber, specificeres betydningen af disse i en semantik, som udtrykkes i et dertil konstrueret specifikationssprog. Der kan nu udføres en fortolkning af begrebsmodellen i henhold til den semantik, som er specificeret. Resultatet er et af mange forskellige præsentationer — eng: *views* — af modellen. Eksempler kunne være, visualisering fra forskellige synsvinkler, beregningsudtryk for stressanalyse, osv.

Relevans for industri og forskning: Inden for forskningsområdet *inkremental design* opfattes design som en proces, hvor objekter, egenskaber for objekter, og relationer mellem objekter tilføjes successivt til en designrepræsentation. I artiklen Chapter 4 argumenterer vi, at værktøjer til begrebsmæssig bygningsmodellering skal understøtte en sådan inkrementalitet samt, at nye navne for egenskaber skal kunne introduceres under designprocessen, efterhånden som disse er nødvendige for at udtrykke den pågældende designidé. Sådanne funktionaliteter understøttes ikke af nuværende designværktøjer.
Denne artikel udgør Chapter 4. Hvis egenskaber skal kunne introduceres løbende, uden for megen restrukturering og omprogrammering

af selve designprogrammet, skal det være muligt at specificere betydningen af de nyintroducerede egenskabers navne separat. Princippet om *semantisk parametriseret fortolkning* udforsker mulighederne for dette.

Udover det teoretiske arbejde er der udviklet et prototypeværktøj, som demonstrerer princippet semantisk parametriseret fortolkning. Værktøjet er programmeret i Standard ML.

1.4.4 Objektaspekter

Reference til fysiske og tænkte objekter som eksempelvis bygninger findes i mange forskellige former for byggedokumentation. Ord og sætninger, som tænkes at referere til sådanne objekter, gør dette på to fronter. Den ene front er i form af de termer, som anvendes. Disse termer kan stå for begreber, under hvilke et objekt hører. Derved er termen med til at karakterisere og identificere objektet, baseret på vores normale forståelse af, hvad termen dækker. Den anden front er at referere til et andet objekt, der står i relation til det pågældende objekt. En vigtig blandt sådanne relationer er relationen mellem del og helhed. De formelt–filosofiske teorier har fællesbetegnelsen *Mereologi*. Imidlertid er Mereologi udsat for en lang række kritikpunkter.

Begrebet objektaspekt introduceres som en lettere form for reference til fysiske objekter. Således introduceres relationen mellem objektaspekter som en mindre forpligtende relation end den i Mereologi. Begrebet defineres gennem en filosofisk diskussion, hvori begrebet *objektaspekt* forsvares imod de vigtigste kritikpunkter af Mereologi.

Relevans for industri og forskning: Arbejdet er et bidrag til forståelsen af de muligheder og begrænsninger, der ligger i forbindelse med dokumentation af fysiske ting såsom bygninger.

1.4.5 Egenskabsteori som fundament for design

Det vises, hvorledes en række filosofiske egenskabsteorier bidrager til forståelsen af begreberne design, designrepræsentation og designrationalitet. Ved designrationalitet forstås her, hvorledes man kan afgøre om et design opfylder en række krav; dvs. om designbeslutninger er rationelle i henhold til kravene. Der søges en filosofisk doktrin, som giver det bedste fundament for at forstå ovenstående

begrebsdannelser. Klassiske teorier om begreber, egenskaber og semantik tages her op til vurdering med designbegrebet som udgangspunkt.

> *Relevans for industri og forskning*: I mange anvendelser af designrepræsentationer og designværktøjer kan man stille sig det spørgsmål, om den viden, som udtrykkes, kan forstås som mere end en række (muligvis underforståede) symboler. Nyere filosofiske egenskabsteorier kæder begrebet egenskab sammen med begrebet kausalitet (dvs. teorier om årsag–virkning). Udfra en række egenskaber kan man ofte sige noget om et objekts mulige dispositioner; dvs. hvad det kan. Eksempelvis kan en bropille modstå en vis vægt, hvis den er opbygget af visse materialer og har visse dimensioner. Man kunne forestille sig, at en sådan viden (som dagligt udtrykkes eller forudsættes i industripraksis og gennem forskning) kan indbygges i datamatbaserede designværktøjer. Herved kunne det blive muligt at udtrække mere information af de designmodeller, som opstilles.

Ens for ovenstående bidrag og derved en gennemgående rød tråd i afhandlingen er, hvorledes forskellige former for viden relaterer, og hvordan den bearbejdes og anvendes som baggrund for dokumentation gennem byggeriets faser. Arbejdet er derved både et studie i hvilke størrelser, som findes og hvordan de relaterer (dvs. ontologi), og hvorledes vi kan anvende sådanne erkendelser som fundamenter for værktøjer og ny teknologi.

1.5 What this work is *not* about

We want to emphasize a number of issues which otherwise might lead to a misunderstanding of how to read this thesis.

Several ideas and concepts — primarily from computer science — are utilized and introduced in this thesis. That, however, does not mean that the thesis is *about* these subjects specifically. They are only to attract attention to the extent that they serve as inspiration and solutions to the problems considered. An example is the notion of Galois connections. The notion has been utilized as a theoretical foundation for relating civil engineering concepts based on set–theory. However, drawing more attention to the theories surrounding the notion — like algebra and advanced lattice theory — may remove focus from the subject in question. We use the concepts in our solutions, but on the individual fields in which the concepts usually belong, we are not experts. This goes for notions like denotational semantics, abstract interpretation, term rewriting, etc. These theories are used in various ways throughout the thesis, but we shall not be concerned with the research areas of these as in computer science.

Similar holds for the approaches of philosophical kind. Here we have a subject of which we are certainly not experts. Therefore, we do not intent to present thorough philosophical arguments which relate to all standard criticism as would be the proper way in philosophy. We do so, only to the extent that our knowledge reaches. The thesis is not a philosophical study, but does take philosophical approaches.

In other words, the thesis may be seen as amateur work on the three fields individually. The professionalism — we believe — appears from our combination of concepts from the three fields. More important, the professionalism comes from seeing the relations in the domain and how to introduce the various concepts for its clarification.

In the area of civil engineering and design, the thesis may be criticized for not considering more empirical examples from civil engineering projects and real designing. Also, it may be criticized for not relating closely to common civil engineering practice on the conceptual front. We believe that we relate strongly on the intrinsic front, but we have made an effort of not founding our analysis and conceptualisation on existing conventions. That might have destroyed our possibility to see things clear and from new angles. In addition, our approach may be criticized for focusing too much on computer science terms. However, recall that this is our aim and origin.

1.6 On the use of "*we*" and "*I*"

In the thesis, we shall follow a general principle and write in first person *pluralis*. This is ordinary praxis in research, and it indicates that scientific achievements usually cannot be credited to just one person. In the case of the present thesis, inspiration, suggestions, and guidance are due to supervisors, researchers, philosophers, and authorities in industry. Also, writing in first person *pluralis* seems to better motivate the reader to feel involved in the presentation which is given.

However, another tradition exists in philosophical writings. Most philosophical literature is written in first person *singularis* or in some special cases as fictional conversations. By writing in first person *singularis*, it is often possible to be more precise in a presentation or discussion. The tradition may be due to a consideration of philosophy as a quest in which contributions appear as ideas and beliefs due to persons as individuals. As individuals, philosophers state their ideas and relate these to the ideas of other philosophers. Thereby, philosophy — as a comprehensive study of foundations — is a conversation between the contributors. For the papers in Chapter 8 and Chapter 9, we have followed the tradition and written in first person *singularis*.

1.7 Reading guide

The thesis is a collection of papers which are presented in individual chapters. After a presentation of related work in Chapter 2 — which may be skimmed or skipped — we present the papers grouped into parts. The parts are named by the overall notion of which the papers are concerned. That is: *Concepts* (Part II), *Design* (Part III) and *Philosophy* (Part IV).

Part II contains Chapter 3, in which we models of two civil engineering concepts and relate these by means of the mathematical notion of *Galois connection*.

Part III covers widely and contains Chapter 4, Chapter 5, Chapter 6, Chapter 7, which all concern design, the process of designing and the foundation for design tools. We go from some early considerations in Chapter 4 which outlines ideas for the notion of *design lattices* and semantic parameterised interpretation. The notion of design lattices is defined and explored in Chapter 5 and Chapter 6. The chapters present two different approaches to defining that same notion and making a mathematical foundation for incremental design. The foundation for incremental design tools is further explored in Chapter 7where we suggest a

new architecture for such tools. This architecture is based on a language– and semantics–oriented approach. The paper in Chapter 4 has been presented on a conference on IT in construction. Therefore, formal aspects are not emphasized in this paper. The papers in Chapter 5 and Chapter 6 consider similar issues. The former has a long introduction to the problem and people who are most interested in the formal aspects may want to skip some of the first sections. In the latter, Section 6.2.2 concerns philosophical issues and can be skipped. The paper in Chapter 7 mixes domain issues, formal aspects and tool considerations. It can be read with focus on various subjects, but the rather long introduction may be elementary.

Part IV contains Chapter 8 and Chapter 9. In the former, we are concerned with the mereological aspects of objects being described as in requirements and design documents. In the latter, we are concerned with three philosophical problems in context of designing. The paper in Chapter 8 is best read thoroughly. The paper in Chapter 9 treats three different design related problems. The sections dealing with these can be read almost independently.

Part V contains a short presentation of a prototype software system for conceptual, incremental design. The implementation is made in order to demonstrate ideas and principles presented in Chapter 7.

Part VI sums up on the overall thesis results in Chapter 11 and presents general ideas for future work in Chapter 12.

Appendix A is a short introduction to The RAISE Specification Language (RSL) which is used throughout the thesis.

Besides reading the papers in the given order, there are two other ways to read the thesis.

In the former, the philosophical papers are read last and thus considered additional to the other papers. Here, we can follow two different paths: An *ontology–orientation* path which focuses on Chapter 3 and a *design–orientation* path which focuses on Chapter 7. Figure 1.1 shows the corresponding dependency graph.

In the latter way, the philosophical papers are read first as prerequisites to the other papers. Again, we can follow the two different paths of *ontology–orientation* and *design–orientation*. Figure 1.2 shows the corresponding dependency graph.

As it appears from the figures, Chapter 2 can be read in any sequence with the other chapters.

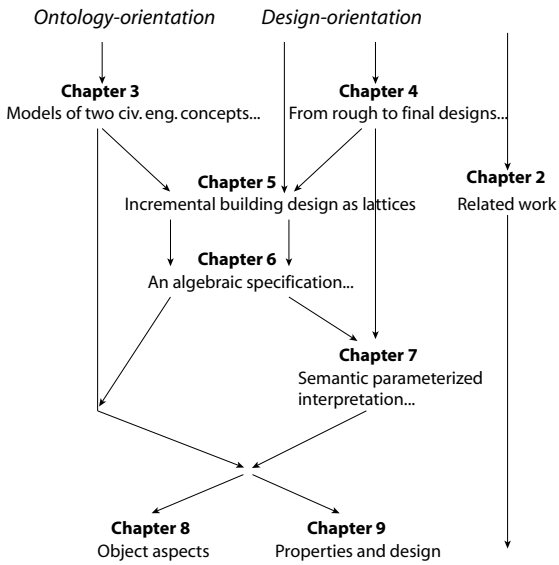


Figure 1.1: Having philosophy as additional.

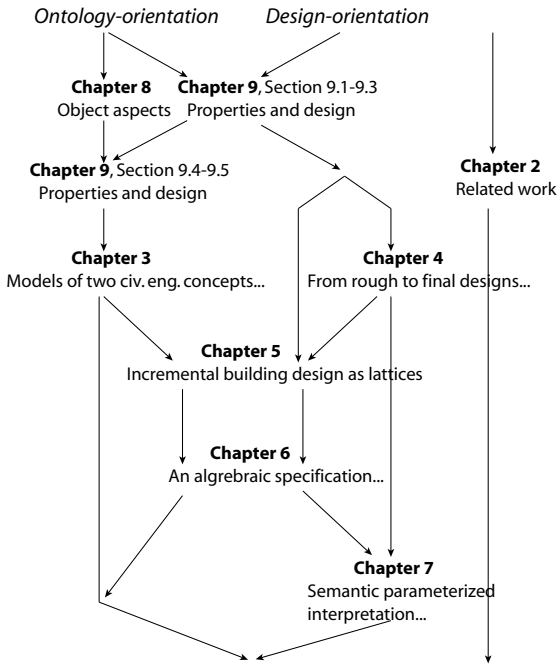


Figure 1.2: Having philosophy as foundation.

Chapter 2

Related work

Research in the area of construction informatics is *fragmented* in the sense that many different approaches to handling building information are explored without following strict guidelines. The reason is that we here have a field without many previous studies. In computer science, we stand on the shoulders of giants in the areas of logic, theory of computation, mathematics, and philosophy. In building, we can base research on a long tradition and on much experience. In the area of construction informatics, this is not so as this field is immature. Thus, there is no specific works which take the same approaches and treat the same subjects as we shall do in this thesis. However, there are several important works which relates to the individual approaches and subjects treated. In the following, we give a broad overview of some of these works.

2.1 Domain engineering

Domain engineering is the activity of establishing models (i.e. descriptions) of "real–world" things and phenomena, and it is considered an important foundation for stating requirements to software systems. Early definitions of the notion of "domains" are due to Arango and Iscoe [4, 104]. Clarification of the rôle of domain engineering and its description based approach is due to Jackson and his distinction between machine and environment [108, 109, 106, 107]. Further

work in the methodology and epistemology of domain engineering as well as its relation to requirements acquisition and software design is due to Bjørner. Arango, Jackson and Bjørner have individually claimed a need for intensive research in the methodology of domain engineering, as well as for exploration and conceptualisation of various domains. The latter, in order for future software systems to be well founded.

The original motivation for domain engineering is that in order to state requirements, a solid understanding of the application domain is necessary. Just as requirements and design of mechanical systems are expressed using terms and rules from physics and chemistry, so may software requirements need to be expressed using terms denoting concepts and objects of the application domain. Requirements and design specifications may be formal and informal, and effort should be made for reducing ambiguity and making the specifications rational. Domains, on the other hand, are informal and may often be irrational. In domains, trains crash, people make design mistakes, and protocols disagree. A description — a model — of a domain should therefore try to capture the essence of the domain; leaving space for phenomena to appear freely as they do.

An initiate step to grasp and find a structure of complexities in domains, is to simply describe them. That is, we need to designate and relate fundamental concepts of the domain. These concepts may represent information to be processed, entities to be described, and agents operating in the domain.

However, besides being a prerequisite to requirements engineering, domain engineering also has a right of its own. Studying a domain conceptually is like when biologists study animals in nature. Through observation, describing, and modelling, the behaviour and characteristics of animals are captured. The established models make it possible — to some extent — to predict behaviour and tendencies.

Models in general have similarities to scientific theories. They contain concepts references and rules for deducing properties and behaviour of the things or phenomena modelled, and of the model itself. In computer science, a study of a domain is an informatic way to establish such a theory. It is so, as mathematical concepts and principles — together with informal (but systematic) methods of describing — are applied in the formulation of the model. That is, domain entities may be modelled as functions, graphs, mappings, sets, lists, etc. A result of a modelling process may be that a business needs to be rearranged (business–process reengineering) or even that no software/hardware system will be of any benefit.

The kernel subject in domain engineering is *how* to acquire knowledge of domain entities, *how* to describe them and *how* to analyse the descriptions. Hence,

with its focus on conceptual modelling, the discipline of domain engineering is intimately related to philosophical notions related to language and semantics.

In domain and requirements engineering, there appears to be at least two distinct paradigms. The first paradigm focus on capturing the basic structures of a domain. Such structures are called "*the intrinsics*" by Bjørner. The second paradigm is rooted in the notion of *goals* and aims at establishing agent–oriented models which capture the individual agendas of agents. It is a paradigm which is only on the sketch board for domain engineering, but has been a front row agenda in requirements acquisition for years.

In both paradigms, capturing the individual perspectives of agents is evident. Still, the two paradigms seem to differ in how much focus these aspects are going to have. In the first, they assist a common structure which strives towards a fundamental core–model which (ever–)lasts. This, however, is not to say that there is always one true and correct model. In the second, the notion of objectivity is degraded in favour of the subjective perspectives and personal agenda of agents. Following the second paradigm to its extreme, we may conclude that the only common structures of a domain appears from negotiation and compromises between agents.

In the following, we present some early perspectives on domain engineering due to Arango. Then we shall present Jackson's and Bjørner's contributions representing the first paradigm, followed by the goal–oriented approach of KAOS representing the second. However, it is important to state that introducing the notion of goals in domain engineering is quite an unexploited idea. Approaches like KAOS relate to requirements engineering. Thus, our discussion of the second paradigm is merely a discussion of prospects for the area of domain engineering.

2.1.1 Arango: some early definitions

Arango and Iscoe consider domain engineering to be indispensable in context of software reuse. According to Arango, it is the desire for reusable and well–tested general software components which calls for a thorough study of an application domain [4, 104, 5].

The central issue of domain engineering is that of *domain analysis*. It comprises the processes of (i) *domain characterisation*, (ii) *data collection*, (iii) *data analysis*, (iv) *classification*, and (v) *domain model evaluation*. The domain characterisation initiates the process of domain analysis. The result is a classification of domain descriptions and via abstraction, a taxonomical concept structure is built together with a vocabulary.

After domain analysis follows the process of *conceptual analysis* and *constructive analysis*. The former aims at identifying suitable concept names with which systems in the domain can be characterised. The latter aims at identifying suitable concept names for implementing such systems. Thus, the former concerns external while the latter internal issues for systems.

The definition of domain engineering outlines a general principle of knowledge acquisition based on conceptualisation. That is, it is based on observation, data collection and analysis. However, it is not clear *how* the moves from stage to stage (e.g. from data collection to classification) are made nor after *what* principles domains are characterised.

2.1.2 Jackson: the machine–environment distinction

Jackson claims that a sharp distinction between domain and requirements must be made in order for specifications in software development to be clear and concise [106, 107, 108]. Usually, we cannot tell from a formal specification whether it is the requirements or design of some system, or whether it describes the environment in which such a system may be introduced.

Jackson draws a line between system related problems and environment related problems, which leads to a definition of the notion of *domain*. Since any kind of system seems to exist for the purpose of interaction with its environment, the environment may be of special interest when acquiring knowledge for the requirements to the system. Jackson uses the term *domain* in order not to degrade its importance. Domain engineering is now a prerequisite to requirements engineering. The essential domain concepts to which requirements specifications may refer, exist independently of whether any system is considered.

In order to acquire proper domain descriptions Jackson's defines the notions of *designation*, *definition* and *refutable assertion*:

Designations with which we name domain concepts. E.g. "*There are* doctors, patients, *and* medical records".

Definitions with which delimitation and conventions are made. Else, we may not be able to capture domain aspects at all. E.g. "*A medical record consists of . . .*"

Refutable assertions which are important as only description being refutable are informative concerning the domain. E.g. "*Illnesses are cured*". If an

assertion about a domain is not refutable — not falsifiable — we have said nothing of importance.

What we strive at in a domain engineering process is to increasingly contribute with knowledge about the domain. Descriptions of such knowledge must be informative which means that they need to carry significant cognitive knowledge. We need this such that we intensively can investigate the border which divides the scopes where a model holds and where it does not.

On the technological side, a domain description $\mathcal{D}$ has the rôle that together with the software design specification and software $\mathcal{S}$ it serves as foundation for judging whether requirements $\mathcal{R}$ are met:

$$\mathcal{S}, \mathcal{D} \vdash \mathcal{R} \tag{2.1}$$

2.1.3 Bjørner: language–orientation

The present thesis draws on Jackson's principles and concepts, but methodologically it is closest to Bjørner's work. Bjørner has contributed to the research on domain engineering by building on Jackson in studying how various domains can be formally modelled in an informatic way. The results are important contributions to software engineering methodology as well as conceptual and ontological clarifications within the areas studied.

The treated domains include the domains of *financial systems* [24], *logistics* [26], *health–care* [25], *railways* [31], *e–commerce* [20, 21], *project, production, planning, monitoring and control* [28], and *air traffic control* [14].

Furthermore, works which in general study the computer science methodology for domain acquisition, includes: [15, 23, 30]. Bjørner has developed a paradigm for systematic, semi–formal software development which goes from domain via requirements to software design. The paradigm is called *TripTych* [16, 22].

When making models, there appears to be two types: *prescriptive models* and *descriptive models*. A prescriptive model is a model which is associated by an attitude indicating the model is normative. A descriptive model is not associated by such an attitude. Bjørner — following Jackson — takes prescriptive models to belong to requirements and software design, whereas descriptive models are the essence of domain engineering. Bjørner makes intensive use of formal specifications in VDM and RSL, whereas Jackson keeps to simple formalisms like

predicate calculus.

The primary focus for Bjørner is the "basic stuff" which he calls the *the intrinsics* of the domain. The intrinsics of the domain of freight and logistics may be that of having goods stored in different places and transporting goods from place to place such that demands and deadlines are met.

Thus, the intrinsics consists of the concepts and structures without which the domain considered would literally fall apart. If we exclude goods and means for transportation, we do not have the domain of freight and logistics anymore. However, we *can* exclude such notions as trains and trucks for transportation, as we may simply carry the goods in our bare hands.

The existence of domain concepts and structures may not depend on who is taking the perspective nor what domain agent to consider. What we strive at is to put on an objective perspective on the domain and simply describe what is going on. Thus, we strive towards the common structures of the domain before looking at tendencies which may be in conflict. At this point, the approach is distinct from goal–oriented approaches (see Section 2.1.4). The intrinsics of a domain is modelled as types, and values of these types can be observed by means of observer functions. Observer functions are the most abstract ways of formally defining recognition rules for domain concepts.

E.g. we may write an observer function which gives the GPS position of some goods, or we may write an observer function which gives the set of travels possible within the restrictions of a time table. Only the signatures of these functions are specified. The former function may in addition be defined in requirements by introducing new technology; the latter may not be possible to define explicitly.

Beside the intrinsics, Bjørner identifies other domain facets like *support technologies*, *rules & regulations*, and *management and organisation* [29].

The domain descriptions lead to *conceptualisation* which tries to answer questions like *what does it mean to have a medical record*, *what is a railway net*, etc? Thereby, we strive towards a clarification of the basic terms and the concepts they denote. The concepts are modelled in formal specification languages; primarily RSL (The RAISE Specification Language[1]) [134, 135]. The connection between the model terms and our understanding is established by various descriptions (rough sketches, narratives, definitions, etc.). Each of these refer to domain model terms. That is, they designate domain concepts.

[1] RAISE stands for Rigorous Approach to Industrial Software Engineering.

One of the important elements in Bjørner's approach to domain modelling is the language–oriented understanding of systems [27]. This understanding puts language and semantics in focus, and the notion of abstract interpretation in front row. The understanding has been of crucial inspiration for the thesis and for our treatment of civil engineering and design.

The language–oriented approach utilizes a special way of modelling which is called *semantic modelling*. It aims at associating certain domain concepts in a denotational way. The meaning (denotation) of a time table could be the set of all possible travels. The value of the time table type is considered a piece of syntax and the semantic values are travel. The interpretation is here abstract, and there may be many such abstract interpretations of time–tables. Each of these present one way of seeing time tables.

The notion of abstract interpretation is here broadened compared to how the notion is understood in areas like program analysis. In program analysis, abstract interpretation is taken to be an interpretation which is alternative to the usual denotational or operational definition of the meaning of language constructs. Instead of defining the meaning of an assignment sentence as a function overriding the variable environment, we may define it as a function adding the variable name to a list of "touched" variables. This list can be used in analysis of the current program; e.g. for optimisation purposes.

However, in the broad picture, we can not really say that one formally defined interpretation is more alternative than another, so we might as well cover them all with the common notion of *abstract interpretation*.

The principle of linking domain concepts by means of abstract interpretation, is the very basis for our principle of relating concepts using the Galois approach (see Chapter 3). Also, it has served as great inspiration to the principle of *semantic parameterized interpretation* (see Chapter 7).

2.1.4 Goal–orientation

The second paradigm is that of goal–orientation which originally focused on requirements acquisition. Ideas now suggest similar approaches for domain acquisition and modelling. Goal–orientation has been a subject in various works on requirements acquisition, including Lamsweerde's KAOS[2] [117, 166], Castro and Mylopoulos' Tropos [44], and Yu's I^* notation [174, 173]. In the following, we shall focus on the KAOS approach.

[2]KAOS is an acronym for *Knowledge Acquisition in autOmated Specification*.

Goal–driven modelling is based on the conviction that goals of agents should govern the elaboration of systems. The notion of goals is mostly used in context of requirements, where intentions of agents are analysed and structured into conjunction and disjunction goal trees [50, 166, 118]. Goals are designated as either supporting or contradicting in a process aiming at evaluating the importance of goals against each other.

Bunge defines the notion of goals in context of problem solving [40]. A goal is an intended state or behaviour of a system. When a goal is an intended state, we must assume that this means a state relatively to some current state. For goals that indicate requirements to a system, the current state does not include the system. A goal of the latter kind is an intended world state in which a system with certain features is introduced, or it is simply a structure change of the world. In either case, we can define the notion of goal as a function from state to state. When a goal is a desired behaviour of a system, we have a similar structure besides that the function from state to state is the denotation of the properties of that system. For a clarification on this subject matter, we refer Chapter 9.

In KAOS, goal trees are used as foundation for assigning responsibilities to agents. Agents can be software components, autonomous software agents, human agents, support technologies, etc. By assigning responsibilities to agents, software architectures for solving overall goals, arise. The idea is that goals in an organisation lead to requirements in the sense that goals explain and justify the requirements to a system. The transition from identification of goals (overall and for each operating agent) to the definition of requirements is performed through a sequence of steps. Each step is governed by three sorts of pragmatic questions: The *What*, *Why*, and *How*. The steps in KAOS are as follows [117, 166]:

1. Acquire goal structure and identify concerned objects.
2. Identify potential agents and their capabilities.
3. Operationalize goals into constraints.
4. Refine objects and actions.
5. Derive strengthened actions and objects to ensure constraints.
6. Identify alternative responsibilities.
7. Assign responsibilities to agents.

The KAOS method has been argued useful in a number of requirements acquisition processes (we refer to papers on goal modelling in general). Its strength, we believe, is not so much the set of precise steps but the idea of assigning responsibilities to agents.

However, it seems that applying a goal–oriented method like KAOS in domain modelling leave us in a dilemma. Focussing on goals makes it difficult to maintain a pure descriptive relationship to the domain as such goals are necessarily captured in a prescriptive way. Forcing goals to be of primary concern and denying there to be a common conceptual structure turns domain modelling into a pure negotiation process between conflicting goals. The basis on which goals are measured in the solution must be rooted in some common domain structure or in the modellers own intentions. In both cases, steps away from the domain are taken.

We did try to model aspects of civil engineering by taking a goal–oriented approach. The scope considered was that of civil engineering projects. Three incomparable perspectives — *quality*, *economics* and *time* — were identified. The identification of these three concepts as three different sorts of values with which to measure project decisions, were hardly any contribution. The solution of finding a domain model took shape of a quite general negotiation process, and the informative, discovering process of domain conceptualisation was lost.

Another problem is that in KAOS there is no step in which domain concepts are designated. Simply, we cannot start with the goals in domain engineering, as these need to be formulated in terms of domain concepts. If we insist in formulating goals for each identified agent, these goals may be formulated in terms which are not cooperative; i.e. different terms may turn out to refer to the same concept, or the same term may have different meanings in different goal formulations. We could follow the terminology by Opdahl et al and call the former *construct redundancy* and the latter *construct overload* [3] [132].

Goal–oriented approaches like KAOS were originally and mainly designed for requirements acquisition. Applying a sharp distinction between requirements (including early requirements) and domain engineering, seems to make it difficult for goal–oriented approaches to succeed in domain engineering. The problem, we believe, is that goal–oriented approaches are highly *prescriptive*, whereas domains can only be captured in a *descriptive* way[4].

Still, we cannot deny the fact that there are two aspects of goals or intensions

[3] However, note that these terms are used in an analysis between ontological concepts and object–oriented concepts in UML. In our case we have a similar relation, but between terms and domain concepts.

[4] according to verbal discussion with Michael Jackson.

in any domain engineering process. The first is that we do domain modelling for a reason. That reason may influence our modelling approach and thus shape the domain model in a certain way. The second is that agents operating in the domain indeed have personal goals, needs, desires, agendas, etc. The question is now whether we should root the domain engineering process in a knowledge–believe–goal orientation, or whether we should try to incorporate agents perspective without letting these overwhelm the common domain structure being modelled. We hold the latter.

2.1.5 Object–orientation

The essence of object–orientation is to work *taxonomically* with data. By taxonomically, we understand that datatypes can be related by a *kind–of* relation[5]. From this notion, the concept of inheritance is derived. To object–orientation is also associated certain language features which facilitate *data–hiding* (encapsulation) and parameterized datatypes. The taxonomical aspect and the notion of parameterized datatypes are dynamic aspects of the semantics, whereas notions like hiding are most often static. The notion of parameterized datatypes has been successfully incorporated in languages which are not usually considered object–oriented. Thus, the taxonomical issue seems to be the best characteristic of object–orientation.

Object–orientation can also be seen as an attempt to simulate the real causal world (or part of it) in the machine[6]. Hence, physical objects and phenomena become objects in a program. However, the analogy does not dictate any use of classes in favour of types. Neither does it dictate any use of objects in favour of values. Often a distinction between objects and values is made. In this sense, objects have names which rigididly distinguish them even though they have the same properties. Values, on the other hand are identical if the meaning of their representations are identical. In this sense, there is only one value 7. However, with abstract types — as in the highly abstract approach in RAISE — it is possible to model physical objects and phenomena simply by using types and values. Bjørner and Haxthausen intensively use abstract types this way; e.g. see [98].

Basically, we see object–orientation as a way of thinking. The more organizational issues of programs are not really of interest in domain modelling. Thus, we shall in this thesis not make such distinctions as between types and classes, or values and objects.

[5] Also called an *is–a* relation.

[6] The first object–oriented programming language was named Simula for this reason [1].

A deep and thorough study of the formal foundations for object–orientation and programming languages has been presented by Abadi and Cardelli [1]. The work goes through basic as well as advanced notions of object–orientation. The aim is to model object–oriented language features by means of mathematical calculi. It is argued that many concepts in object–oriented languages root in similar mathematical abstractions. Modelling object–oriented language features by means of fundamental calculi, may clarify such issues.

Both functional calculi and object calculi are presented, although the focus is on object calculi. The object calculi take the notion of objects as primitive, whereas the functional calculi take the notion of function as primitive. The calculi are presented in order of increasing complexity starting with the untyped object calculus ς and the corresponding functional λ–calculus. The most advanced object calculus presented is $\mathbf{Ob}_{\omega<:\mu}$ which provides features like subtyping, variance, quantified types and type operations. The most advanced functional calculus presented is $\mathbf{F}_{<:\mu}$.

The papers in Chapter 5 and Chapter 6 utilize the fundamental understanding of objects as records which underlies most of the treatments by Abadi and Cardelli. Furthermore, a notion like subsumption — formally defined in [1] — is used in Chapter 5, Chapter 6 and Chapter 7, but we assume anti–symmetry to hold as is the case for *sub–classing is sub–typing*.

2.1.6 A few words on UML

A trend in the object–oriented area suggests modelling using the Unified Modelling Language (UML) [33]. In the present thesis, The RAISE Specification Language (RSL[7]) is primarily used as we find this language suitable for the designation process applied. It would be messy to use the graphical notation of UML together with the domain descriptions which designate domain concepts. The result would necessarily be a separation of UML diagrams and domain descriptions, in which case the idea of applying one modelling language seems lost.

A number of problems with UML has been stated. E.g. Opdahl and Henderson–Sellers claim that the ontological basis of UML has certain problems of ambiguity and inconsistency [132]. Another, more general problem is that domains are not always object–oriented. The way we have approached the domain of civil engineering and design, is not of taxonomical kind. A large number of such approaches have already been taken. Many of these are of technological and

[7] A short introduction to RSL is given in Appendix A.

conventional kind and does not contribute with any real ontological clarification. Examples are the classification systems of SfB, CBC, BSAB 96 and IFC (see Section 2.2.4.1–2.2.4.4).

Another reason for not using UML is that language and semantic aspect are simply not nicely expressed in UML, and for notions like well–formed predicates, we need to use a formal language like OCL[8] anyway.

2.1.7 Other domain studies

Beside the domain studies of Bjørner, several other works have applied RSL or similar languages for modelling.

An interesting study of RSL and RAISE in domain modelling can be found in the case study [165]. Among the cases, we found the specification of spatial concepts relevant to space requirements in civil engineering. Also, the case study which models object–orientation and design patterns is interesting as it shows the strength and generality of RSL.

Furthermore, RSL has been used to model railways and public transportation by Satchok [146], and verification and security systems of railways by Haxthausen and Peleska [98].

2.2 Relating civil engineering concepts

One of the main contributions of the thesis is the approach of relating domain concepts by means of Galois connections. Thus, related work to this subject matter includes other Galois approaches in computer science, other approaches for relating civil engineering concepts, and classification systems in the construction industry. A large number of contributions fit into the second category, but we shall concentrate on Galle's notion of artefaction and Ekholm's conceptual treatments.

[8] OCL stands for *Object Constraint Language*.

2.2.1 Galois connections applied

The notion of Galois connections is a general mathematical notion which is defined as pair of dual, monotonously decreasing mappings [85].

Between a set of objects and a set of properties there is a Galois connection which binds sets of objects to their common properties and sets of properties to the sets of objects possessing them. The connection hierarchically orders object sets and property sets and thus has been the foundation for theories of types and classes in computer science. The orders define complete lattices [13]. Thus, the notion of Galois connections and the notion of lattices are fundamental to the definition of programming languages [161], Formal Concept Analysis [85], object–oriented modelling [124], and knowledge representation [158]. Other work which relate to the notion of lattices, includes work on the foundation for knowledge representation [131, 129] and connections to databases [128]. General principles of and foundations for conceptual modelling are presented in [158, 48]. More deep ontological considerations focus on notions like identity [92, 91, 94, 93].

Philosophically, the notion of Galois connections are rooted in the idea that some objects have common properties and that this common "thing" is a meta–physical entity. This idea is also known as the idea of "*one–over–many*" (see [143] or Section 9.2.2.1 in Chapter 9).

The notion of Galois connections gives the mathematical means for classifying objects according to their kinds. This way, a set of objects is partitioned into powersets which are closed under set–inclusion. Similar goes for properties. A hierarchical structure appears from set–inclusion of objects and the ordering spans a complete lattice with Top represented by the empty set and bottom represented by the full set being partitioned. A dual lattice from the ordering of set–inclusion of properties will always exists.

Haav has worked with the classification possibilities offered by Galois connections and lattices [96, 130]. The work shows how dynamic classification can be done without breaking the order consistence of the lattice.

The notion of Galois connections has also been applied in areas which do not directly relate to knowledge representation and classification. An example is Ingleby's attempt to reduce the complexity of model–checking of railway control and security systems [103]. By assigning properties to railway lines, a Galois connection is defined. The connection is instantiated and is represented as a matrix (a formal context) where rows list routes of the railway and columns list line segments. The application of the Galois connection is now as follows. From

a formal context stating which routes overlap which lines, a classification can be made [85]. The result is a complete lattice in which the Top is the set of all routes and the bottom is the empty set of routes. Analysing the lattice, we can identify *complementary* sets of routes. Two sets are complementary if and only if no node in the lattice represents a set of routes which contains routes from both, and which is not the empty set.

Model–checking is a task which is likely to explode in complexity. However, the complexity can be reduced by making a partition of the problem at hand. The Galois connection and the analysis described above makes it possible to split up the space of combinations.

Other work which utilizes or explores the application of Galois connections, include approaches to automatic concept formation [95, 101],

2.2.2 Galle's notion of artefaction

Galle has described the process of "*artefaction*" as a process of communication going through the stages of *briefing*, *designing* and *making*. Through stages of development, building ideas are expressed and interpreted by agents working constructively in the domain. Representations include sketches, CAD–drawings, database schemes, and the results of interpretations lead to productions of revised representations, representations belonging to the next stages of development (like from requirements to design), and the artefact itself. The representations are bound together by means of what Galle calls a *relevant successor*. A given design is a relevant successor of a design brief if an only if the agent who has specified the design brief considers the design to "contain" what has been expressed in the design brief. For the relation between design and requirements, we may say that the relevant successor — as a predicate — states that the design satisfies the requirements. However, it is crucial to Galle, that such judgement cannot always be based on universal agreement. Rather, the communication process involves interpretation of symbols, and the notion of relevant successor is defined ranging over the ideas which are conceived by agents performing the interpretation. For a more detailed presentation and for a description of how the work of this thesis relates to Galle's work in this field, we refer to Section 2.3.2.3.

2.2.3 Ekholm's treatments

Ekholm has worked intensively with the foundation for conceptual building design [64, 67, 68, 69] and his work has been published in relation to classification

projects in building as well as in relation to the BAS•CAAD project [66, 65]. The work is special in that it aims at approaching conceptual building modelling on the basis of well defined philosophical and ontological notions, mostly due to Bunge [40, 38].

The works include clarifications on the theoretical foundation for BSAB 96, as well as studies on construction concepts like *work* [64] and *space* [69]. Central in these studies are the notions of objects, properties, activity, and semantics.

2.2.4 Classification systems in building

We shall consider a number of the most important classification systems: *SfB*, *CBC*, *BSAB*, and *IFC*. However, we shall not here consider the important work of Gielingh, as this work is described in Section 5.2.1, Chapter 5.

2.2.4.1 SfB

SfB stands for *Samarbetskommittén for Byggnadsfrågor* (eng: *Co–ordination Committee for the Building Trade*[9]). The commitee has contributed to a paradigm shift in classification of elements, processes, and information in the building sector [90].

In the middle of the 1940's, a confusion in construction information and management, was recognized. This confusion was due to the lack of generally accepted codes for identifying building elements, processes, and information. New paradigms and patterns of trade were replacing old ones. SfB noticed that trade could not be the main class in future classification of elements, processes, and information. Basically, the sharp borders between disciplines like HVAC, Electrics, etc. were becoming "blurred', and a classification paradigm which could accommodate cross–discipline knowledge were needed. This led to a focus on a distinction between "*what*" and "*how*".

The result of a construction process is a building component; i.e. a part of a building or a complete building. However, such components may not depend on "*how*" they are constructed. Thereby, a dichotomy between *building elements* ("*what*") and *construction works* ("*how*") was introduced. Also, an additional class of material was introduced. This class served as a sub-division of construction works.

[9] Thanks are due to Prof. Anders Ekholm for clarifying this issue.

Three tables of concepts were developed and each concept was represented by a code: Large letters for construction works, small letters in combination with numbers represented material, and numbers in parentheses represented building elements. E.g. "Ce4(21)" represents the concept of *ground works with sandstone for external walls*. "C" stands for ground work, "e4" stands for the material of sandstone, and "(21)" stands for external walls.

It is essential that the notion of materials is related as a sub–class to the notion of construction works and not elements. Material is usually an important factor in the identification of a construction work; e.g. *wood work* and *steel construction*.

Even though the SfB system appears to be simple and not conceptually sophisticated, its development today stands as one of the major break through in the conceptualisation of construction and construction information. It does so basically by its distinction between process and product. The way of combining two incomparable classes — one of construction work and one of building element — gives a certain flexibility. The fact that we combine two incomparable concepts means that we are in fact performing a kind multiple inheritance, though only between the two incomparable classes. However, that principle may be applied thoroughly as suggested by the CBC and in Chapter 4, Chapter 5 and Chapter 6.

Another contribution worth mentioning is that the SfB facilitates multiple view (multiple interpretations) on the same code. Since classification and codes are not grouped by trade; the building owner, the architect, and the constructor have the freedom to put individual "eyes" on the same code. This freedom has been utilized in CAD programs where the SfB coding system has been used intensively as foundation for layering systems. A layering system — in CAD — splits up drawing information using filters.

However, the SfB system does not specify any well–formed conditions for what combinations of the three tables make sense. The reason may be that it is defined for *non–exclusive* classification and is not rooted in natural laws or Boolean algebra. Simply, if an element does not fit into any given category, a new category can be created. This process characterises the way in which the categories in SfB have been developed.

Also, it is clear that SfB and variants have been developed with a focus on syntax for the representation of codes — a focus the computer science and informatics disciplines have left long ago. In computer science, it is a growing opinion that focus in modelling should be on the semantic values and not on syntactical nor representational [27].

2.2.4.2 CBC

The CBC — mostly due to Bindslev (a presentation of CBC is given in [90]) — draws intensively on the work and experience from SfB. CBC stands for *Co–ordinated Building Communication*. Bindslev argues that set theory and Boolean algebra should be the mathematical foundation for building classification system. That is, the *ad hoc* way of classification in SfB is not satisfactory.

To Bindslev it was non–sense that a code like "Ff" arbitrarily could mean different things depending on in what context it was interpreted. A more strict approach — focusing only on two different ways of interpreting — was the result.

CBS — as SfB — emphasises the dichotomy of "*what*" and "*how*" in classification and in coding. Each double code represents a class of "activity types". Each type is a set of activities which produce the same sort of building element and by the same sort of construction. The types are not further partitioned into sub–classes but additional numbers are taken to do this job. The numbers are introduced *ad hoc* in order not to restrict the necessary distinctions of activities to be defined in a standard set of classes. As in SfB, different interpretations can be made of the same code. However, in CBC, the only two ways of interpreting codes, exist: *as activity* and *as result* .

The main contribution of the CBC is its arrangement of concepts within each of the tables. This arrangement strives towards a real classification based on Boolean algebra, although it does not put effort on multiple inheritance. As in SfB, CBC identifies three facets of building: *elements*, *construction* and *resources*. This is the viewpoint of the architect; the reverse order is the viewpoint of the contractors.

The perspective seems logical, but also unnecessary complicated from a mereological point of view. We may simplify things by saying that elements consist of parts and that each part is an element as well. We thus end up with the mathematical founded and flexible notion of *bill–of–material* as in [61, 17, 18].

Also, the distinction between element and resources is context–dependent. From the view point of the supplier, a pre–cast concrete wall is an element to be produced, whereas from the view point of the contractor on the building site, it is a resource. We believe that a more flexible perspective — hence, better principles for classification — can be found in Chapter 3 and Chapter 5.

2.2.4.3 BSAB 96

The BSAB 96 system is a successor of the SfB and CBC systems, although it is considered the "*not–SfB system*". The system is an approach to classification as well as system definition of buildings, and it is theoretically rooted in a more thorough study of meta–physics than its predecessors [35].

BSAB 96 defines a number of main categories for classification:

construction complex. By a construction complex is understood a "*a collection of adjacent construction entities, serving one or more user activities or functions*".

construction entity. By a construction entity is understood "*an artefact, permanently attached to the ground, which independently enables a user activity or function*".

element. By an element is understood "*a part of a construction entity, which fulfills a predominating function in the construction entity.*

designed element. By a designed element is understood an "*element*" *seen* as something providing functionality to the surroundings. In BSAB terms: "*a technical solution of an element*".

work result. By a work result is understood an "*element*" *seen* as the physical result of production or maintenance. In BSAB terms: "*a result of an activity on the construction site for the production of [a] part of or a whole construction entity*".

space. By a space is understood "*a three dimensional material construction result which can be used for a certain purpose and has a defined extension within, or in connection to a construction entity*".

resource. By a resource is understood the things or services consumed in a construction activity. Examples are *manpower*, *construction products*, *machines*, etc.

construction product. By a construction product is understood "*a product intended for incorporation in a construction entity*".

The definition of construction entity as something attached to the ground we believe is weak. It is hardly a sufficient criteria, and the distinction between entity and complex also seems to be confusing. A similar problem of ambiguity seems to exist between the notions of construction entity and element. Also,

notice that the notion of *construction product*, being a resource, is not defined as a kind of element.

Common to SfB, CBC and BSAB 96 is a kind of confusion between the rôles played by parts in a building. The confusion appears in BSAB 96 in that several concepts seem to denote the same; namely the concept of a physical thing having spatial extension. In BSAB, we have the distinctions between *construction complexes*, *construction elements*, and *elements*. However, we believe that this distinction is made on weak ontological grounds; primarily for mereological reasons. Certainly, a construction complex like a university consists of a collection of buildings, and certainly these buildings consist of walls, ceilings, doors, etc. But where exactly should we draw the distinctions? A building being part of a university complex may as well be considered a construction complex. Even a single exterior wall can be considered a construction complex as it consists of parts which again consist of parts, etc. The confusion, we believe, is rooted in two problems. The first is the mereological problem of universally making a distinction between parts and wholes conceptually. This issue has been discussed in Chapter 8. The second problem is that the hierarchy of parts (from construction complexes to elements) is not defined recursively, as suggested in Chapter 5.

2.2.4.4 IFC

The *Industrial Foundation Classes* (*IFC*) is a conceptual framework and classification system aimed for achieving interoperability between different software applications in different industries. A technical presentation of IFC is given in [123]. Discussions on application of IFC in Sweden is given by Ekholm et al in [70]. The IFC is developed and maintained by the *International Alliance of Interoperability* (*IAI*). This organisation out springs from AutoDesk©.

The IFC is a collection of base classes divided into four layers. The idea is to offer a collection of standard classes to which applications can refer, and thereby get a common basis for interoperability. The idea is distinct from that of founding all applications on the same conceptual structure. Each application can have its own conceptual structure and then "derive" common classes from IFC. These classes can be specialised within the current application following the common principles of class inheritance in object–orientation. The four layers of IFC are: the *resource* layer, the *core* layer, the *interoperability* layer, and the *domain* layer.

The resource layer offers a set of classes which are considered to be general characteristics of objects in industrial software applications. It includes classes

for *geometry*, *actors*, *date* and *time*, *measures*, etc.

The core layer offers classes which are fundamental for knowledge representation like *objects*, *properties*, and *relations*. The three classes have one superclass which is called *the Root*. The Root contains attributes for storing general administrative information like *owner history*, *name*, and *descriptions*. The object class is specialised into the classes of: *product*, *control*, *actor*, *project*, *process*, and *resource*. The resource class is, however, not to be confused with the resource layer; nor with the classes on this layer. The sub–classes of the object class are further specialised and decomposed.

The interoperability layer serves as connection point between applications of different disciplines and working differently with data of equal kinds.

The domain layer can be seen as a layer on which base classes of the other layers are specialised to fit conventions of a certain application domain. Application domains include *HVAC*, *Electrical*, *Architecture*, or *Facilities Management*.

IFC has been subject to various criticism. Ekholm states that IFC is established on a foundation which lacks of basic philosophy [70]. In general IFC lacks of precise definitions of the individual classes, and each class seems too general to be of much use. It is argued that IFC focus too much on CAD application even though notions like actors and project are included. Also, it is not clear how the interoperability principle is going to work. As an example, Ekholm states that the notion of property sets are included in IFC. However, there is no specification of how to introduce new kinds of properties and how these are to be linked to the IFC class of property sets, if this can be done at all within the framework of IFC.

We shall, however, emphasize some other issues on which we find IFC weak and not satisfactory as a framework for our study of civil engineering and design. The first issue is that IFC does not seem to be founded on a sound ontological and philosophical basis. This reflects in some inconsistencies and confusion on terminology. In the following, we list (in random order) some of these.

- Defining notions like properties and relations as classes introduce a confusion as it is then uncertain how to understand the instances of these classes. Are they objects?, and in what sense do they differ from instances of the object class? Also, what is the connection between instances of a property class and the attributes of the object class? Ontologically, they both aim at characterising the object in question.
- The class of properties are partitioned into several sub–classes. One is for so–called *single value properties*; another is for so–called *enumerated*

value properties. It seems like a structuring of properties based on Boolean algebra might clarify this distinction by applying restrictions on the value sets of a property. This is to some extend the approach taken in Chapter 5, Chapter 6, and Chapter 7.

- IFC makes a distinction between products, actors, and resources. However, this distinction seems to be conventional rather than ontological. Basically, all instances of the three classes are resources; hence, the class of products and actors might as well have the resource class as super–class.
- Furthermore, the generality of the class definitions yields trouble for the class of decomposition. Decomposition is considered a sub–class of relations. However, there is not restriction which permits cycles.
- It seems like we with IFC have a lot of classes and that many of these have been introduced when they have been needed. Thus, the hierarchy of classes seems not to be founded on considerations of the ontological commitments the classes induce. E.g. we have five different sorts of relations. Among these, we have a definition–relations which is hardly relations in the ontological sense.

The second issue is that IFC defines the relations between concepts (fundamental like geometry as well as domain specific) *by convention*. Concepts are not related because they can be justified to satisfy ontological rules like subsumption, or similar rules over the structure of concept models. Thereby, the conceptual system becomes more a political and conventional contribution than a technological one. In Chapter 3, we have outlined a way of relating domain models using a semantic approach. The approach is based on the notion of Galois connections which mathematically bind models of concepts together. The predicate on which a Galois connection is based specifies *how* the two concepts relate. Applying this kind of analysis — rooted in good modelling methods, in mathematics, and in philosophy — we believe would benefit the IFC.

2.2.4.5 Other

Other works on product models and frameworks include the STEP approach [6], the application of the language Prolog in design and manufacturing [56], and studies of the dynamics of data in civil engineering and design [57].

Furthermore, it is worth mentioning studies of the foundation for conformance checking [52], conceptualisation of civil engineering with computational perspectives by Rezgui, Cooper, Björk, Froese, and Paulson [138, 162], and approaches

in general for coupling civil engineering project information by Turk, Björk, et al [163, 32, 89].

2.3 Design

Our work on design relates to philosophical, operational and representational issues. In the following, we shall consider related work in these areas.

Furthermore, general concepts and principles for systematic and successful requirements and design acquisition, and communication has been presented by Cherry [45] and Salisbury [145].

2.3.1 Philosophy and fundamentals

In this thesis, we have partly turned philosophy of design into philosophy of language and metaphysics. Thus, an actual survey on design from a philosophical perspective, and with contributors on the field of design, has been limited works by Schön, Galle, Ekholm, and Alexander.

Work which indirectly has been of inspiration, includes Gärdenfors' notion of conceptual spaces [86] and various selected sections in the works of Bunge [40, 38, 39, 37, 36].

2.3.1.1 Schön

Schön's work stands as one of the main contributions to design and architectural thinking. The work of Schön spans broad and covers practical issues, epistemological aspects as well as issues relevant to the philosophical area of scientific discovery. In some sense, his notion of *seeing–as*, has served as inspiration to the papers in Chapter 7 and Chapter 9. Still, the connection to Schön is in these papers is quite loose. Schön's work, definitions, principles, and perspectives have influenced many design researchers. One is Valkenburg and Dorst's analysis of design teams [164].

Schön favours a perspective on design as a process of communication; not a target–oriented problem solving process [149]. In traditional problem solving, all information needed is present from the beginning. The process of reaching a solution (e.g. a design configuration) can be described as variable assignments

such that the design constraints — derived from the requirements — are satisfied. Schön rejects this as a proper understanding of most design processes. It is hardly a fruitful approach when designing complex artefacts like buildings as the number of constraints and dependencies may be huge, and some constraints even impossible to define explicitly. Also, we may not succeed in approaching a design in a strictly compositional way. What might seem badly at first may, taking another view, turn out to contribute positively to the solution.

The classical Petra–Smith conversation illustrates this. Through iterations of design elaboration, a solution is approached. The essence is this. We need to focus on one aspect (a discipline) and work that through; ignoring its relation to the surroundings and other constraints. We then rewind and start over again by putting on another perspective or taking up another discipline. The process, Schön says, is a kind of conversation with the situation. This conversation aims at contributing to the awareness of problems and possible solutions in each iteration. Schön also calls it a *reflective conversation with the situation*. It is reflective because the design is iteratively revised based on ideas conceived when reviewing the design iteratively.

The knowledge of the practitioner is not entirely something prerequisite to the design process. There are knowledge which is conceived during the reflection process and there are knowledge which is tacit. By tacit knowledge, we understand knowledge of which we may be unaware or which may be difficult to describe or explain. An example is the knowledge of how to ride a bike.

A practitioner like a designer makes many judgements about the quality and correctness of representations, products, and decisions. But it may be without any clear and explicable criteria. And the practitioner may display skills of which there are no explicitly defined algorithms. We act on the situation as we reflect on that situation and the actions we are performing: *reflection in action*.

An example of reflection in action is jazz musicians playing and improvising. Sometimes reflection makes them collectively increase in volume or tempo; other times they get inspiration from each other to turn the music in certain directions. Through listening they feel where the music is going. Knowing the music score intimately, its variations, and their own abilities, makes it possible to perform a kind of *top–down* approach, but along the way things happen, which require adjustments to the situation in a *bottom–up* manner.

The idea of focusing on a discipline has been adopted (modelled) in the notion of design lattices (see Chapter 5) where a design process can go in various directions which can be combined.

2.3.1.2 Alexander's patterns

Alexander's contributions to the area of design, and especially architectural design, has been tremendous. We shall not cover all aspects of his contributions but merely focus on the notion which mostly relates to the work in this thesis — the notion of *patterns* [3].

Alexander is concerned with the structure of buildings — not the mereological aspects — but the intentional perspective on objects and the many problems that arise in that area. He recognizes that the world — and especially the world of architecture — is filled with things repeating again and again. Curves, dimensions, combinations of materials, etc. are like patterns that we see everyday. If every church is different in the sense that they occupy distinct extensions in space, how come are they all considered to be churches. Certainly, notions like use, rôle and conventions may be part of the definition of what it means for something to be a church, but such notions may be insufficient for proper and precise definitions. There may be such things as objective and universal patterns which repeat themselves in things we consider to be alike; an epistemological perspective shared with realists on properties like Russell. At this point, Alexander's considerations relates to our study of properties in design (see Chapter 9).

The notion of pattern is defined by *"morphological laws* of the form:

$$X \rightarrow r(A_1, A_1, \ldots)$$

The law reads: in a context of type X, between the elements A_i there is a relation r. Recursion is now introduced such that a law itself can be a pattern. Thereby, Alexander avoids having to separate elements from relations.

On defining patterns explicitly, three things need to be identified: the sorts (the *what*), the pragmatics (the *why*) and placement in space and time (the *where* and *when*).

Here, Alexander works quite intentionally and suggests questions to be asked in the reflection process. Such questions could be: "*What is solved by putting a door here*?", or "*what does it for a room to raise the ceiling*?", etc. One of the examples considers a living room which on the one hand must satisfy the requirement of letting members of the family be together, and on the other hand allow members to work separately on their individual hobbies and being able to leave things from day to day. Finding a solution to this design problem needs clarification on the various needs.

The solution defines a room with a number of large alcoves. The solution is an example of a pattern which can be applied whenever the mentioned intentional forces are present. The work [3] is Alexander's, Ishikawa's, and Silverstein's suggestions to a series of patterns going from development of towns to design of interieur.

The idea of patterns — even though normative — seems similar to the notion of intrinsics in domain engineering (see Section 2.1.3). More precisely, our models of domain intrinsics may follow Alexander patterns as certain concepts are best modelled as graphs and other as sets. Furthermore, the definition of a semantics for interpreting conceptual design models — as suggested in Chapter 7 also brings the notion of patterns to mind. What appears again and again may not be the various semantics, but the specification of sorts due to the limitation in language (see argumentation in Chapter 7).

2.3.1.3 Galle

Galle often claims a commitment to Platonism in context of design [81, 83]. However, as argued in Chapter 8 and Chapter 9, a possible worlds perspective is often just as good and in some cases better. Galle identifies an ontological dilemma when it comes to the notions of design and design representation: What are the entities that designers describe and what do they mean? The question is called *the problem of the absent artefact* and is defined by Galle in [80]. We treat this problem in Chapter 9.

The ontological space suggested by Galle defines a dichotomy between abstract entities and concrete entities. The abstract space is divided into that of concepts and ideas (platonic objects). Concrete entities include such objects as physical buildings, digital design databases, drawings, scale models, etc. Concrete entities have abstract counterparts. The link between abstract and concrete space is established by a two way relation. From abstract to concrete, we have the relation (or the act) of *production*. From concrete to abstract, we have the relation (or the act) of *interpretation*. It is essential in Galle's ontology, that the result of an interpretation is an abstract entity. Often in computer science we use the notion of interpretation (or abstract interpretation) to mean the process of interpreting as well as the representational result of such a process. In this sense, a document can be considered an interpretation of another document. To Galle, that would be comprising of three actions: interpretation (giving an abstract entity), getting the meaning (as an act going from one abstract entity to another), and production (projecting the latter abstract entity into the causal space).

A design model like a drawing has a meaning as interpreting it yields an abstract counterpart. The meaning of this counterpart is the counterpart of an artefact. This artefact may or may not exist physically, but it can be brought into being by means of production. The abstract counterparts of physical entities fall under concepts which are abstract as well. E.g., we have the concepts of drawings, drawing databases, etc. There may be a whole hierarchy of such concepts. The counterparts are said to be *examples* of drawings, drawing databases, etc. The concepts of drawings, drawing databases, etc. are rooted in the common concept of models. In a sense, we may say that model–concepts are propositions. The abstract counterparts of physical entities like buildings do not fall under the concept of models. Here, Galle makes a distinction — a distinction, we try to break in Chapter 9. The abstract counterpart of a certain building, falls under the concept of buildings, and the concept of buildings is a sub–type of the concept of artefacts.

2.3.1.4 Ekholm

Ekholm's work has influenced theoretical as well as practical areas of design; especially the areas of classification and the ontological foundations for design tools. As in other areas, Ekholm's work is rooted in the philosophy and ontology of Bunge. Ekholm draws — as Bunge — a sharp distinction between two kinds of properties: Intrinsic properties and extrinsic properties. The former are the kind of properties which are solely connected to the object. These are the properties an object possess independently of any relation it may have to the environment. The latter are the kind of properties an object has in virtue of its relation to its environment. Ekholm suggests that extrinsic properties are what makes an object offer certain functionality. Thus, the extrinsic properties are interesting when modelling the interface between the object and its environment. Contrary, intrinsic properties concern the internal structure of an object; e.g. shape and material.

A system — like a designed artefact — is a collection of parts composed in a certain way. The functional view on a system emphasizes the extrinsic properties and thus the systems ability to interact with an environment in certain ways. The compositional view emphasizes the intrinsic properties, the parts, and the internal structure of the system.

In the thesis, we have followed the ontology of Shoemaker in excluding extrinsic properties in favour of relations. We believe that in fact such notions as shape and material properties are what makes an object offer certain functionality. The relation to which an object stands to its environment is — besides its spatio–temporal position — to be seen as the causal dispositions possessed by

the object. A more detailed study on this subject can be found in Chapter 9.

2.3.2 Operational issues of design

The operational issues of design are centred on the notion of design moves as a means for going from stage to stage. In the following, we consider related work contributing to the clarification of the design process and of the cognitive and concrete moves performed in these.

2.3.2.1 Schön's notion of design move

The changes made by the design practitioner when *in action*, Schön calls the *moves*. A move may be sketching something, starting all over again, etc. In our thesis, we have adopted Schön's notion of *design moves* as those made in design processes. Where our view significantly differ from Schön's is in our aim of defining such moves explicitly. This aim is, however, not to be seen as an attempt to restrict the free reflective process of designing. The aim is to capture the moves being made and the elaboration of designs in a systematic way. Still, we endeavour that the designer is aware of the moves being made which is a view supported by Schön when he focus on the problem naming stage and the reflection aspects of design.

In addition, Schön argues that the collection of moves spans a kind of web with many branches. This idea seems compliant with what we call *design lattices* in Chapter 5.

2.3.2.2 Ekholm's principle of incrementality

Ekholm understands design as a problem solving process. A problem — following Bunge — is defined as "[*a*] *lack of solution knowledge*" [40]. The design process is considered an iterative process. From the formulation of a problem, we identify the interface between the environment and a potential system. Via synthesis, a tentative design solution is sketched and through analysis it is verified against the problem at hand. Synthesis may be regarded as going from a functional view to a compositional view, while analysis may be regarded as going the opposed way. The analysis aims at identifying the properties of the system needed. The result of the analysis is added to the knowledge already present and further iterations can be made based on a revised problem definition. The

analysis also indicates whether a satisfactory design solution has been reached in which case no more iterations are needed. This cyclic process is also called the *Generator–Test Cycle* by Simon [154].

The principle of incrementality is fundamental to Ekholm's understanding of design and it is a principle that to a large extent has been adopted for this thesis.

The paper presented in Chapter 4 is the result of collaboration with Ekholm. The collaboration also motivated for the elaboration of ideas for the kind of conceptual design systems outlined in Chapter 7.

2.3.2.3 Galle's notion of artefaction

Galle considers — as Schön — the design process to be a process of communication. He defines the notion of *artefaction* which is a process of specifying, designing, and producing artefacts [79]. An artefaction process goes through stages in which ideas and solutions are communicated between agents. Self–communication is here a special case.

For artefaction to be successful is for the communication to be successful. To illustrate this, Galle has made a generic model of communication between two agents, and stated a condition for successful communication.

The model relies heavily on what Galle calls a *relevant successor* denoted $c_1 \Rightarrow c_2$. It is a two–argument predicate which applies if and only if whatever a certain agent interprets from c_1 it will also interpret from c_2. In this sense, it is a special version of implication ranging over representations being interpreted by agents, or simply over agent ideas. We shall here override $\Rightarrow$ for both. The communication between an agent A and an agent B can now be described as follows[10]:

1. A conceives an idea $\mathcal{I}_{\mathcal{A}}$ and expresses this idea to an agent B by a symbol s_1.
2. B interprets the symbol s_1 and conceives an idea $\mathcal{I}_{\mathcal{B}}$.
3. B conceives another idea $\mathcal{I}'_{\mathcal{B}}$ which (to B) is a relevant successor of $\mathcal{I}_{\mathcal{B}}$.
4. B expresses the idea $\mathcal{I}'_{\mathcal{B}}$ to A by a symbol s_2.

[10] The number of steps can be reduced if A=B.

5. A conceives an idea $\mathcal{I}'_{\mathcal{A}}$ according to the symbol s_2.

The communication is successful if and only if A considers $\mathcal{I}'_{\mathcal{A}}$ a relevant successor of $\mathcal{I}_{\mathcal{A}}$.

Figure 2.1 is similar to the figure given in [79], except that we state two possible failure situations instead of one.

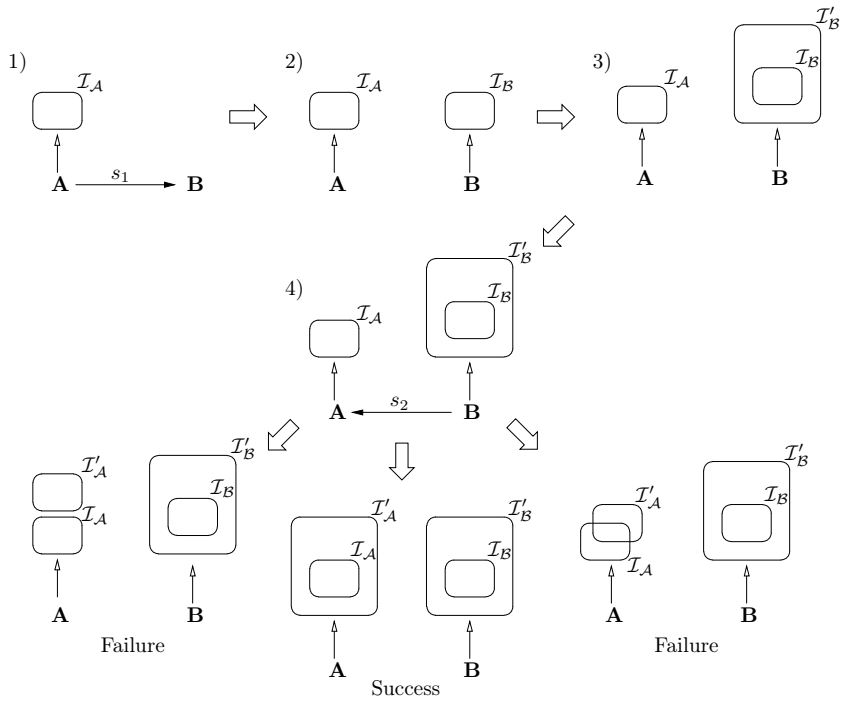


Figure 2.1: Communication Model

The model is instantiated into a part of the domain of civil engineering by introducing three types of agents: *the client*, *the designer*, and *the maker*. The process of artefaction now goes as follows. The client expresses a need in a design brief. The brief is interpreted by the designer who makes a design. If the design is a relevant successor of the design brief, this communication is successful. The design is further interpreted by the maker, and so on.

The essence of the model is that between representations, there are relevant bindings which relate knowledge and things through stages of building projects.

Galle's notion of relevant successor and the model of communication has been an important inspiration source for the development of the notion of design lattices. In such lattices, a partial order between design representations is explicitly and formally defined. The design representations are called *artefact models* and correspond to the bubbles in Galle's diagram.

However, we had to freely interpret how to understand the bubbles in the diagram. The diagram was meant to be a pictorial intuitive presentation, not a very formal one. Bringing on the formal "machinery", demands an answer to the question of what the bubbles in the diagram represent. It cannot be objects as in a possible worlds semantics. That would lead to the mistake that a set of objects is a relevant successor of another if the former subsumes the latter. The implication must go the other way. The Galois connection between objects and common properties, reversing the ordering, calls for understanding the bubbles as sets of properties of an object. This is a consistent model, but it is insufficient if several individual objects are considered part of the same artefact. A solution is to introduce representations which comprise both taxonomical and part–whole information as is the case with *artefact models* presented in Chapter 5, Chapter 6 and Chapter 7.

2.3.2.4 Operationalisation of needs

Alexander and Poyner make some fundamental definitions concerning needs in the process of building design [2]. The foundation is the idea that there are no such thing as objective truth and that it therefore is not possible to write an objective and "correct" programme which yields some actual physical geometry of the building in mind.

The elaboration of a design program must start by identifying the needs of the users of the building. The problem is now that the notion of needs is not a well defined concept. Alexander and Poyner therefore replace the seemingly static notion of needs with an operational notion of *tendencies*. By a tendency they understand something which users would usually do, like moving closer to the window or turning up the heat. Tendencies can be tested by means of observation; needs cannot. A similar observation turns goals into intrinsics in domain engineering (see the discussion of the two paradigms in Section 2.1.3 and 2.1.4).

The design problem — the programme — is now to find a physical environment which meets the needs in shape of tendencies. The aim of that environment, and thus the aim of introducing a system like a building, is to open up for the possibility that people can act as they like. There will always be multiple

conflicting needs but to Alexander and Poyner these are due to the conditions under which the needs are considered. In general, they believe that there can always be a design which solves all conflicts without putting on priorities. At this point, we disagree. To us, design includes defining priorities in order to solve certain unavoidable conflicts.

In both cases, conflicts are solved by introducing geometrical or topological relations between objects. We begin with the empty set of relations (assuming no conflicts) and then add relations alternately in order to solve the conflicts when discovered. However, we only want to state a minimal set of such relations. Adding more relations than necessary may yield unnecessary restrictions.

The design approach which is presented is — concerning the incremental approach — similar to the one taken in Chapter 5 and Chapter 6. We share the idea of starting with the empty configuration, although we have not explicitly incorporated the notion of needs nor tendencies in our work. In essence, there is no concept of activity or observation of behaviour in the models presented in the thesis. In this sense, we have not as Alexander and Poyner aimed at modelling the design process in its intrinsics, but in its representational and step–wise aspects.

2.3.2.5 Refinement in software engineering

The notion of refinement in systematic software specification and development has a special connection to the ideas of building design and design lattices presented in this thesis. The connection is the analogy of relating specifications in an order–theoretic way.

The overall idea of refinement is as follows. We consider specification languages to be special programming languages having features which are not all computable. These features makes it possible to easily express requirements and designs of software systems. However, in order to reach a software specification which can be implemented, the use of the special features need to be eliminated. Thus, the use of the features need to be substituted with corresponding computational constructs.

The process can be done step–wise and is called *refinement* [9, 10, 126]. An underlying calculus is called a *refinement calculus*. A refinement of a statement S into a corresponding statement S' via a command α should satisfy certain conditions to ensure semantic consistency. We write $S \leq_\alpha S'$ to denote that S' is a refinement of S. This means that whatever S means, so does S'. In practice, the partial order is applied on predicates that applies to S and S'. Thus, the

partial order on predicates ($P \leq Q$) is logical implication ($P \Rightarrow Q$).

The refinement principle and the partial order $\leq$ is quite similar to the principle of design lattices, and the partial order of design stages, presented in Chapter 5 and Chapter 6. However, we take the ordering symbol $\leq$ to apply in the opposed way as we take $a \leq b$ to mean that a is more specialised than b because the space of interpretation is restricted.

In refinement, a predicate P usually states the weakest pre–condition. That is, *at least* P must hold after the execution of a statement from the initial state. A similar philosophy founds the notion of design lattices and the notion of lattices is present in research on software refinement as well [168]. Each design stage ascribes a certain set of properties, relations, objects, and decompositions, to a model. These can be seen as predicates ranging over artefacts (abstract or concrete) which satisfy the criteria of having (at least) these properties, relations, objects and decompositions.

2.3.2.6 Other

In addition, several other works provide important perspectives on the process of design and on design moves. A prominent of these is Roozenburg's and Eekels' definition of the design process as an iterative process involving sketches and evaluation according to values [139]. The desire for certain values is the reason for making changes. Such values could be to relax by sailing on the ocean in weekends. The means for doing so is what Roozenburg and Eekels calls the function. Sailing on the ocean requires a boat, and certain properties are then necessary for reaching this functionality for an artefact. These are the properties which make it possible for an artefact to sail, and from the properties we derive a certain form for the artefact. Thus, the process of designing is a goal–oriented process of reasoning going from values in life via function and properties to form.

2.3.3 Representational issues of design

The representational issues of design are centred on how to represent models of artefacts. We shall consider approaches for object representations, the notion of complex objects, and core–models aimed at facilitating smooth communication and knowledge sharing.

2.3.3.1 Objects as records

A common model of objects is to consider these as records of attribute–value pairs. This model has been suggested and thorougly explored by Cardelli [42]. The model reduces the complexity of handling objects to a minimum. Furthermore, it emphasizes the notion of subsumption of types, and orders object types in lattice structures. The notion of subsumption is important to the notion of design lattices as presented in Chapter 5. It is so in the sense that it is the mathematical foundation for incrementally ascribing properties to objects. The notion of subsumption is defined in Chapter 7 as well as in [1]. Also, the notion of multiple inheritance has been of inspiration to the papers in Chapter 5 and Chapter 6. The notion has in these papers been carried out thoroughly in the sense that the type–structures of objects form complete lattices. Furthermore, the understanding of objects as records has been borrowed for the design of the modelling language presented in Chapter 7.

However, the general model of objects as records does not facilitate a distinction between properties that are intrinsic and properties that extrinsic. Neither does it facilitate a distinction between taxonomic information and part–whole information. Thus, attribute–value pairs are used in the same way for representing colour information of cars and the wheels of cars. The three ontologically distinct sorts of information are not distinguished by the notation. In Chapter 6, we have made effort making such a distinction, extrinsic properties are excluded in favour of relations. A discussion of the distinction between intrinsic and extrinsic properties is also given in context of *the problem of prediction* in Chapter 9.

2.3.3.2 Complex objects

Many different approaches have been taken to represent objects in a mathematical way. In focus is the problem of representing and handling what is called *complex objects*. A complex object is an object which may be composed by several other objects. That is, complex objects introduce a part–whole relation. The problem is that the attractive lattice representation — which is useful for representing taxonomic class relations — may not be attractive if also part–whole relations are to be incorporated.

Bancilhon and Khoshafian have suggested a calculus for complex objects which puts a focus on part–whole representation in favour of taxonomic relations [12]. Objects are defined recursively as records of attribute–value pairs where values are objects too. An object can also be a single value of a base type like integer.

A partial ordering is now defined on object in the way that an object x_1 is a sub–object of an object x_2 if and only if x_2 (at least) has the attribute–value pairs of x_1. The definition of partial order is defined recursively over the hierarchical structure of objects.

The approach is fundamentally distinct from ours (see e.g. Chapter 6) and from Cardelli's. The difference is that for complex objects the order is reversed such that having more attribute–value pairs and sub–objects means less specialised. That is, $[a : 1, b : 3] \leq [a : 1, b : 3, c : 4]$ holds. This leaves us with a dilemma concerning specialisation. The problem, we believe appears from structuring according to taxonomic and part–whole information in the same lattice. Separating values of properties from objects solves this problem, as shown in Chapter 6.

2.3.3.3 Core models and knowledge sharing

The notion of *core models* is an approach to product modelling which is rooted in the idea that various perspectives and applications of data need a tidy and common conceptual structure. Various works built more or less on the notion of core models. The problem of having common models is central to that of knowledge sharing.

Important work on product modelling and foundations for data exchange has been done by Eastman [60].

Hendricx has recognised that computers usually do not come into play in design until the late stages [99]. At these stages, most of the design of the building in question has been completed. The work of Hendricx aims at establishing a so–called *core*–model which facilitates design through the various stages of design. The model facilitates that design can be split up i abstractions suitable to the designer. Thus, we may have a *masterplan* level in which buildings are simply solid blocks; we may have a *type* level in which rooms and spaces are identified; etc. At each level, the practitioner can use the sort of information he or she is familiar with. The aim of the underlying core–model is to "glue" all pieces of information together; thereby, relating the stages consistently. An inspiration for the work has here been de Waard's contribution of conformance checking in civil engineering and design [52]. The work uses the methodology of M.E.R.O.D.E (*Model–driven Entity–Relationship Object–oriented Development* [53]).

Ekholm, Fridqvist, and van Leeuwen identify the need for reducing the number of concepts such that context free design systems can be made. They draw on

ideas from Hendricx, and Fridqvist's doctoral thesis on the BAS•CAAD system is one example of an application (see Section 2.4.3).

Galle rejects that design communication is to be based on a large, complex, international classification standard. Rather formal notations close to the way we use natural language should be used [82]. Jacobsen's doctoral thesis suggests that small inter–application models may solve such problems as consistency when different applications have to manage different aspects of designs or other building information [110]. The models, we understand as point–to–point protocols which maintain consistency of building information by a set of updating and translation rules.

2.4 Design tool considerations

The science of computer aided design (CAD) is the study of how to express forms and features with finite sets of characteristics. As a sub–problem, we have the problem of visualising designs using a finite set of properties. The science of CAD emerged from this sub–problem but has expanded as a science to a conceptual level. At this level, models of artefacts are objects to which properties are ascribed and behaviour defined.

2.4.1 Turk

Turk has worked intensively in the interdisciplinary area in which computer science and civil engineering merge. The study is broad and range over areas like conceptual product modelling [170], philosophy, and the foundations for computer aided design (CAD) [169]. Technological aspects and clarifications on standards are also part of the study [163].

In the study of CAD foundations, Turk takes the position that there are problems in considering language as the foundation for exchange of information. In [169] a version of *hermeneutic constructivism* is analysed. The perspective suggests that language is a form for social behaviour and that the meaning of expressions and statements can only be established if the context of these are considered. The problem with CAD and data exchange is — it is argued — that justifying the correctness of conceptual models as we do not have an objective reference against which they can be verified. Similar considerations — but with a different conclusion — are presented in Chapter 4 and Chapter 7.

Furthermore, Turk claims that static type structures, as in object–oriented CAD systems, restrict the practitioner and imply a sort of blindness for alternative design solutions. To this we agree, and the paper in Chapter 7 aims at establishing a new foundation for design systems free of such static structures.

2.4.2 Hakim and Garrett

Maher and Hakim have investigated the distinction between class–centered and object–centered approaches in conceptual modelling of buildings [97, 87]. The object–oriented paradigm is highly *class–centered* which makes it inadequate for handling the dynamical aspects of incremental design. Instead, Hakim and Garrett suggest an *object–centered* approach which is free of static class systems. Implementation is sketched using the description language KL–ONE.

The distinction between class–centered and object–centered modelling has greatly influenced our work on foundations for incremental design; primarily the work presented in Chapter 4 and Chapter 7. However, we recognize in Chapter 7 that a model using an object–centered approach lacks of semantics as the type system is weak. Therefore, a semantics which defines the meaning of property and relation terms, is needed in addition to the model.

2.4.3 BAS•CAAD

The BAS•CAAD[11] system is a prototype design tool for conceptual design, aimed at covering all stages of the design process. Its kernel is a small collection of base types from which generic objects can be made. To these objects, properties (intrinsic and extrinsic) can be ascribed incrementally. In a sense, this means that the BAS•CAAD system breaks the barrier between objects and classes.

BAS•CAAD is centred on the notion of *ThingClass* which is a class of generic objects; i.e. objects to which any set of properties and relations may be ascribed. It is recognized that in order for a design system to offer certain kinds of dynamics, it must be free of what is called *contexts*. That is, it must be independent of what kinds of artefacts we usually design. Such a context is often reflected in the taxonomic structure established for design systems. The dynamics can be only be achieved if we move notions like properties and relations to a dynamic instance level. That is, properties and relations are considered values

[11] BAS•CAAD is an acronym for "*Building and User Activity Systems Modelling for Computer Aided Architectural Design*".

and not static parts of class definitions. Only then can the system support the design process which may search for new ways of understanding the universe of discourse — a primary goal of designing.

The principle of moving properties and relations to a dynamic level is by chance similar to the pre–study for the paper in Chapter 4 [62]. The similarities of ideas with those of BAS•CAAD, became clear when working on this paper. The paper in Chapter 7 goes further and studies the foundation for making design tools completely free of contexts and in which conceptual design models can be given different interpretations.

The BAS•CAAD system has been developed as a prototype by Fridqvist as part of his doctoral thesis [75]. The theoretical foundations are due to Ekholms studies of concepts in civil engineering and design; primarily [67, 66, 63], which also study the incrementality of design processes. In addition, expiriences with BAS•CAAD have been documented [65].

2.4.4 Partial evaluation and compiler generators

Partial evaluation and program transformation have had a tremendous impact on computer science. Thorough presentations on theory and practice include [113]. The first step was to construct interpreters being programs for evaluating (i.e. run) other programs according to some input data. The next step was to construct compilers being programs able of translating a programs written in a source language S into program written in a target language T. The Futamura projections show the correspondence between source and target, and they show how compiler generators can be the result of partial evaluation. A program *mix* performs the mixed computation by interpreting the source language code (s:S) according to input data (d:D). It turns out that the level of compiler generators is a fix–point. This means that the *mix* program and its semantics are the only things needed for making target programs, compilers, and compiler generators. The Futamura projections are (see [76] and [77]):

$$[\![\, [\![\, \text{mix} \,]\!]\ \text{P}\ \text{D} \,]\!] S = [\![\, \text{P} \,]\!]\ D\ S \tag{2.2}$$

$$T = [\![\, \text{mix} \,]\!]\ I\ S \tag{2.3}$$

$$C = [\![\, \text{mix} \,]\!]\ mix\ I \tag{2.4}$$

$$CG = [\![\, \text{mix} \,]\!]\ mix\ mix \tag{2.5}$$

The theory of partial evaluation and compiler generators induced the idea of

whether a similar *mix* program could be made for design systems. It was necessary then to clearly split the design language from the language specifying the meaning of model terms. Combined with the aims of establishing a formal foundation for incremental design systems, the idea of programs for generating design programs lead to the idea of *Semantic parameterized interpretation* presented in Chapter 7. Input data are here design representations on a conceptual form. Interpretation of such models can be made according to a semantics and the result is a target — a view of the model. In a sense, the software architecture suggested in the paper, corresponds to a design system generator. However, effort has not been put into showing that this can be a *mix* program generating itself. The focus has been on accommodating the requirements for design systems supporting incremental design.

2.5 Philosophy

The two papers of philosophical kind — Chapter 8 and Chapter 9 — serve themselves as surveys of related philosophical work. In addition to the philosophical theories treated in these papers, a number of other theories are related to the work of this thesis in general.

Examples are the notion of necessary connection in causation and the broad area of philosophy of language. The main issue in the latter is the foundation for learning and communication, and it covers areas like use–theories of meaning, tacit knowledge, how we can acquire knowledge of new language, and how we can learn to use languages correctly. We shall not mention any contributions here as we necessarily would have to leave out important ones. However, Chapter 9 serves as a good introduction to the philosophical theories which relate to our study of the domain of civil engineering and design.

Part II

Concepts

Chapter 3

Models of two civil engineering concepts and their Galois connection

Abstract: Civil engineers operate with such notions as tree–structured cost frames and graph–structured project plans. We show how a cost frame denotes a possibly infinite set of project plans and, vice–versa, how a project plan denotes a possibly infinite set of cost frames.

The civil engineer, we claim, operates freely in a domain where cost frames determine project plans, namely the project plans which can be executed within the financial restrictions defined by the given cost frame, and vice–versa.

In this paper, we model this two–way connection and show that it forms a Galois connection. We thus believe that our contribution, in its mathematics, captures an everyday concern of civil engineers in planning, management, and control of building projects.

3.1 Introduction

In the domain of civil engineering, much of the knowledge necessary for constructing a building is stored and managed as a collection of documents. If we insist in storing and managing such knowledge by computer systems, it is essential to talk of documents which express building knowledge. Each document contains a description of a certain aspect the building organisation, the processes, or the artefact (the building) to be built. Even though documents contain different kinds of knowledge, they may relate, as the knowledge expressed in one document is necessary for expressing knowledge in another.

When we talk of semantic relations, we usually mean a relation between terms and what such terms stand for; what they denote. It seems, though, that there exists another sort of semantic relation. This relation stands between values of certain domain concepts. An example is the relation between financial documents like budgets and management documents like project plans.

Here, it is important to state that the field of civil engineering can be vague concerning terminology[1]. The notion of *budget* is often used to mean a hierarchical structure of expense designations. However, it may also include revenues, estimated profits, etc. When the information concerns only expenses, the term *cost breakdown* is used. The concept often consists of the sub–concepts of *unit costs* (*concerning material*) and *unit rates* (concerning *work*[2]). However, these names are not uniform. The meaning of the term *cost breakdown* may be misleading. It is not just the costs as numbers which are divided but also the items, which are expanded to sub–structures of the financial frame. Therefore, we suggest the terms *cost frame* instead of *cost breakdown*, and *cost item* to designate the individual items of expenses. That is, cost frames group cost items. If a cost frame is part of a larger cost frame in the hierarchical structure, we shall refer to it as a *sub–frame*. In the domain of civil engineering, the decision whether to break down a cost item is a matter of convention and convenience.

Intuitively, a cost frame and a project plan relate to each other: We can check whether the project plan can be executed within the cost frame, and we can check whether a cost frame covers the expenses for resources which are consumed by operations in a project plan.

Preben Scheutz, architect, writes (translated from Danish, [147]):

[1] Thanks are due to Prof. Stephen Emmitt for clarifying these terms.

[2] In the contract documents for *The Øresund Link* [137], the term applied is: "*Schedule of Unit Rates*". We thank Leif Sjøgren, The Øresund Consortium, for providing the documents.

> "*To define the cost frame and adhere to it through all phases, requires that project solutions are verified according to the cost frame, using a partition into well–arranged cost items.*"

The epistemological question which is sketched is thus: *How do the expressed knowledge of a cost frame contribute to the knowledge necessary for making a project plan*?

This paper presents formal models of two civil engineering concepts and show how a semantic relation between their values can be defined. We consider the concepts of *cost frame* and *project plan*. A generalisation of the relation maps between sets of values of the two concepts. We show that the maps defining this relation explicitly, satisfies Galois criterion[3].

Relating two concepts by a Galois connection means that the concepts are related by the mathematical structures which models them. This is a stronger and more rigour relation than a relation only holding between the names of the concepts.

Our argumentation for bringing on the notion of Galois connections goes as follows. Often we define that certain concepts relate without explicitly specifying the pragmatics behind the relation. That is, the concepts relate *by convention* and we can then only assume that the relation is rooted in the intrinsics — i.e. the basic and fundamental structures — of the domain considered. Relating two concepts based on a Galois connection has the benefit that the relation which is claimed is also justified in set–theory. This means that we from values of the one concept can predicate values of the other and vice versa.

Satifying Galois criterion implies a number of mathematical properties. By expressing the relation in its mathematics, we believe to capture an everyday concern in civil engineering project planning and management. Furthermore, the relation makes the foundation for software systems supporting such activities. In more strong words: If we cannot make this relation explicit, we have no chance of rationalising such civil engineering activities; especially not to introduce computer based supporting tools.

We use The RAISE Specification Language (RSL) to express the formal models, their relations and conditions [134, 135]. However, effort has been made to informally express intuitions behind the relation and the models of the concepts considered.

[3] See the axioms of Definition 3.5.

3.2 Domain concepts

In the following sections, we model the two domain concepts; *cost frame* and *project plan*.

3.2.1 Cost frames

A cost frame (cf:CF) is a mapping from cost items (ci:CI) to a pair of which the first component is the cost (cost:Cost) of that cost item and the second is the sub–frame. A sub–frame is a cost frame as well. Each cost item in a cost frame (including sub–frames) uniquely identifies a record of expenses concerning work, material, man–hour payment, etc. That is, it designates the maximum amount of money that is allowed to be spend. From a cost item and cost frame, we can observe the corresponding frame name (fn:Fn) which is the name it is given in the same cost frame. Such labels are, however, not unique but can appear several places in the same cost frame.

A cost frame is wellformed if and only if each cost is non–zero (we assume that costs are not negative) and that it is equal the sum of costs in the corresponding sub–frames. Furthermore, it is required that all cost items are indeed unique (i.e. the number of cost items is equal the length of the list of frame names). Also, all sub–frames must be wellformed.

```
type
  CF′ = CI m→ (Cost × CF′),
  CF = {|cf:CF′ • wf(cf)|},
  CI,
  Cost

value
  zero : Cost,

  wf: CF′ → Bool
  wf(cf) ≡
    (∀ ci:CI • ci ∈ dom cf ⇒
      let (cst,cf′)=cf(ci) in
        non_zero(cst) ∧
        cst=total(cf′) ∧ wf(cf′)
      end) ∧
    card frames(cf) = len frame_names(cf),
```

```
  frames: CF′ → CI-set
  frames(cf) ≡
    if cf = [ ] then {}
    else
      let ci:CI • ci ∈ dom cf,
                  (cst,cf′)=cf(ci) in
        {ci} ∪ frames(cf \ {ci}) ∪ frames(cf′)
      end
    end,

  frame_names: CF′ → Fn*
  frame_names(cf) ≡
    if cf = [ ] then ⟨⟩
      else
        let ci:CI • ci ∈ dom cf,
                    (cst,cf′)=cf(ci) in
          ⟨obs_Fn_Frm(ci)⟩ ⌢ frame_names(cf \ {ci}) ⌢
            frame_names(cf′)
        end
    end,

  total: CF → Cost
  total(cf) ≡
    if cf = [ ] then zero
    else
      let ci:CI • ci ∈ dom cf,
          (cst,_)=cf(ci) in
        cst + total(cf \ {ci})
      end
    end,

  +: Cost × Cost → Cost,
  =: Cost × Cost → Bool,
  non_zero: Cost → Bool,

  obs_Fn_Frm: CI × CF → Fn,

axiom ∀ c,c′,c″:Cost •
  [ addition ]
  c+c′ ≡ c′+c,
  c+(c′+c″) ≡ (c+c′)+c″,

  [ zero element ]
```

zero+c ≡ c,

The functions iterating on sets are exhaustive because all sets are finite and elements are iteratively removed.

In empirical material like [137] we recognize the pattern that a cost frame divides into sub–structures which are also cost frames.

Let f_{ijk} range over values of CI, and c_{ij} range over values of Cost. Values of CF thus have the general form (here only expanding part of it):

$$\left[\begin{array}{lll}
 & & \\
f_1 & \mapsto & \left(c_1, \left[\begin{array}{lll} f_{11} & \mapsto & (c_{11}, [\,]) \\ f_{12} & \mapsto & (c_{12}, [\,]) \\ \vdots & & \\ f_{1n} & \mapsto & \left(c_{1n}, \left[\begin{array}{lll} f_{1n1} & \mapsto & (c_{1n1}, [\,]) \\ f_{1n2} & \mapsto & (c_{1n2}, [\,]) \\ \vdots & & \\ f_{1nm} & \mapsto & (c_{1nm}, [\,]) \end{array}\right]\right) \end{array}\right]\right) \\
f_2 & \mapsto & (c_2, [\,]) \\
\vdots & & \\
f_p & \mapsto & (c_p, [\,])
\end{array}\right]$$

An example of a value of type CF is:

Example 1

```
value
  workitems, foundations, exc_limestone, exc_quaternary, stone_bed,
  underbase_grouting, backfill, concrete, tremied,
  foundation_in_situ_reinforced, foundation_precast_reinforced,
  ballast_fill, piers_pylons, pp_in_situ_reinforced,
  pp_precast_reinforced, labouritems : CI,

  cf : CF =
    [ workitems ↦ (20005300,
        [ foundations ↦ (17964800,
            [ exc_limestone ↦ (2000000,[ ]),
              exc_quaternary ↦ (400000,[ ]),
```

stone_bed ↦ (1080000,[ ]),
underbase_grouting ↦ (44800,[ ]),
backfill ↦ (400000,[ ]),
concrete ↦ (12200000,
[tremied ↦ (1200000,[ ]),
in_situ_reinforced ↦ (3000000,[ ]),
precast_reinforced ↦ (8000000,[ ])]),
ballast_fill ↦ (1840000,[ ])]),
piers_pylons ↦ (2040500,
[concrete ↦ (2040500,
[in_situ_reinforced ↦ (185500,[ ]),
precast_reinforced ↦ (1855000,[ ])])])]),
labouritems ↦ 6700500]

◇

3.2.2 Object aspects

We introduce the notion of *object aspects* (x:X). These are different from building components and material in the sense that an object aspect (x:X) denotes a certain aspect of an artefact to be built. In this paper it is not considered something consumed by an operation. By an object aspect, we understand a reference to a proper part of an object existing in a possible world equal or succeeding to the actual world. However, in this paper the notion of possible worlds is not important.

Examples of object aspects are: a certain wall, the collection of all exterior walls in a building, the foundation, and the top surface of the foundation. Object aspects are referred to in construction specifications, building codes, conceptual models, requirements, and (perhaps most important) in general talk, e.g. at the construction site. We introduce this notion and distinguish it from physical resources consumed by operations. Thereby, we claim to avoid certain mereological problems. In this paper, we shall not dig into the mereological aspects here, but refer to Chapter 8 which presents a study of the notion.

3.2.3 Resources

Resource types (rn:Rn) denote resources which are consumed by operations. Resources can be components like pre–cast concrete walls or materials like sand.

The amount of resources are measured and represented by natural numbers. Furthermore, resources can be personnel and tools. We do not, however, apply a principle of tool reuse (usage with replacement). The way such resources are managed in most civil engineering projects opens up for a solution, which is different from the obvious computer science one. This obvious solution would be to order operations based on their input–output types. This ways, all components "flow" through the project plan. However, there are a number of problems in this solution which mostly root in some mereological problems of which we shall not be concerned here.

Instead, what is measured for personnel and tools are the hours of work. Personnel and tools are considered resources but are *counted* as *man–hour* and *use–per–hour*, respectively. After an operation has been performed, such resources *has* been consumed as it is the time of use that counts. A personnel resource is counted as a multiplication of the number of persons and the hours of use. Similar goes for tools. Thereby, we avoid optimisation issues and we are still true to the domain.

Definition 3.1 For an operation to "concern" an object aspect, we understand that the operation is or includes the construction of or the work on that object aspect.

□

We assume that resources consumed by operations concerning one object aspect are distinct from those consumed by operations[4] concerning other object aspects. The types and numbers of resources may be the same but the physical resources are not.

The distinction comes from the definition of which resources are consumed in what operations. Let x_i be object aspects. On Figure 3.1, the concrete resources for casting x_2 and x_3 are distinct resources from the resources consumed by operations involving x_1; e.g., paint for painting x_1. Similar, the resources for preparing the surfaces denoted by x_4 are distinct from the resources consumed by the operation which casts x_4. Also, the latter set of resources do not include the former set of resources.

This trick of *not* having set–based resource inclusion downwards the order of object aspects, solves the problem of uniqueness when quantifying over these.

[4] possibly the same.

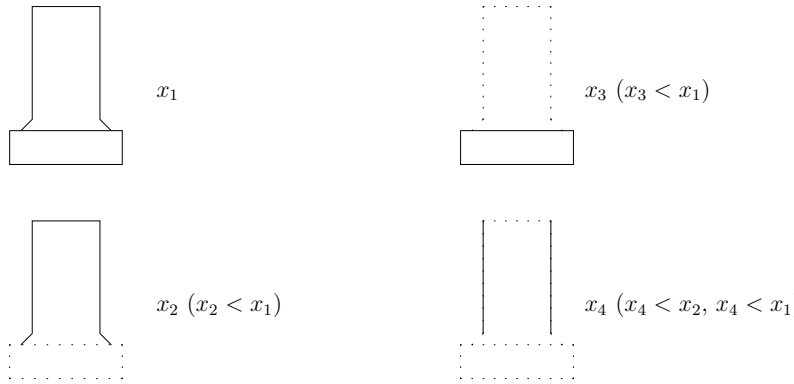
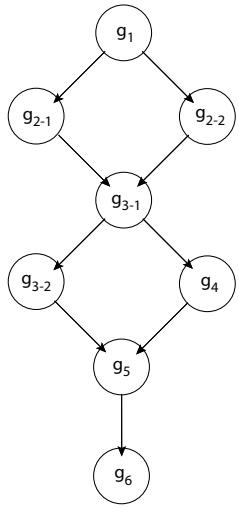


Figure 3.1: Aspects of a bridge pier object.

3.2.4 Project plans

By a project plan (pp:PP) we understand a directed, acyclic graph (DAG) in which nodes (g:Γ) correspond to operations and the set of edges defines a partial ordering on operations.



g$_1$) Foundation molding (x_1)
g$_{2-i}$) Piers mounting ($x_2 = \{x_3, x_4\}$)
g$_{3-j}$) Slabs mounting ($x_5 = \{x_6, x_7\}$)
g$_4$) Structural steel arms ($x_8 = \{x_9, x_{10}\}$)
g$_5$) Pylons mounting ($x_{11} = \{x_{12}, x_{13}\}$)
g$_6$) Slabs waterproofing ($x_{14} < x_5$)

Figure 3.2: Project plan.

Figure 3.2 illustrates a project plan where the nodes are named g_i and x_i names

object aspects. E.g. g_6 names the node in which the waterproofing operation on both slabs is performed. This operation thus concerns the object aspect, which is the top surfaces of both slabs (named x_{14}).

We have applied the same symbol $<$ for orderings of *object aspects* as used for part–whole relations by Simons [155]. $x < y$ — used illustratively in figure texts only — means that x is an object is a proper part of the object of which y is an aspect. E.g. this holds for y being a pier and x being the surface of that pier. Figure 3.3 shows the high bridge of The Øresund Link in section view, and Figure 3.4 shows an aspect partition on such a bridge section.

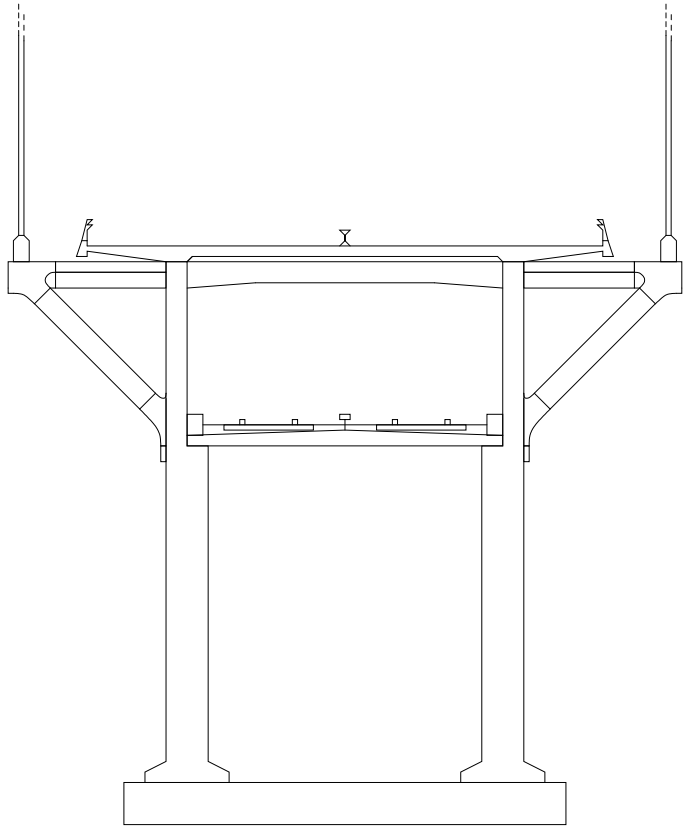


Figure 3.3: The Øresund High Bridge in section view.

From a node in a project plan we can observe the associated type (on:On) of the operation to be performed, as well as the concerned object aspects (xs:X-**set**). A project plan is wellformed if and only if it satisfies the criteria of being a directed acyclic graph.

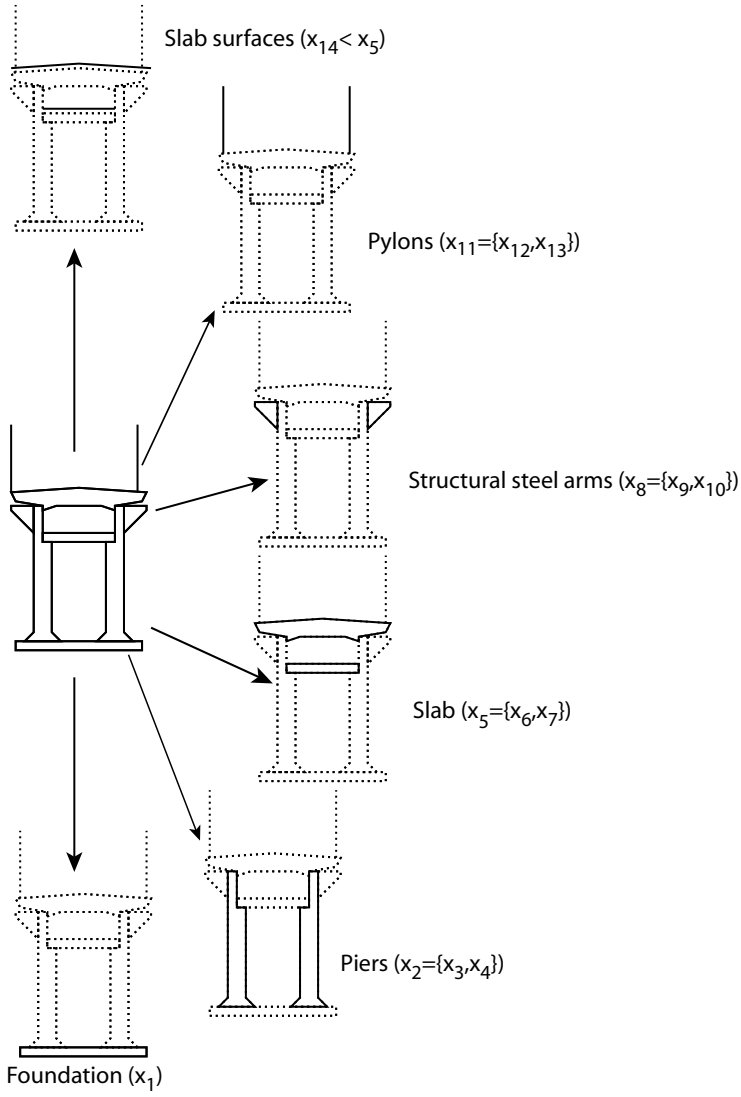


Figure 3.4: Aspects of The High Bridge.

type
PP′ = Γ $\xrightarrow[m]{}$ Γ-**set**,
PP = {|pp:PP′ • wf(pp)|}
Γ,
On,
X

```
value
  obs_On_Γ: Γ → On,
  obs_Xs_Γ: Γ → X-set,

  wf: PP → Bool
  wf(pp) ≡
    (∀ gs:Γ-set • gs ∈ rng pp ⇒ gs ⊆ dom pp) ∧
    (∀ g:Γ • g ∈ dom pp ⇒
      ∼ is_before(g,g_succ)(pp)),

  is_before: Γ × Γ → PP → Bool
  is_before(g,g′)(pp) ≡
    g′ ∈ pp(g) ∨
    (∃ g″:Γ • g″ ∈ pp(g) ∧ is_before(g″,g′)(pp))
```

In the model of project plans, we have not expressed how nor when resources (humans or machines) are or can be available. Such scheduling issues are not directly related to cost frames. However, they are important issues in civil engineering when involving concepts like *manning tables* and *resource allocation*. In this case the model of project plans must be extended.

3.3 Mediating ties

In order to relate cost frames to project plans, we present models of additional concepts. These represent the information (often assumed or being background knowledge) necessary for making project plans and verifying these financially. We call these additional concepts *mediating ties*[5].

The mediating ties, relevant for relating cost frames and project plans are the notion of *price index* and the notion of *work index*. A price index (prcidx:PrcIdx) maps resource types to a unit cost function (ucm:UCF). A unit cost function map units of measure (u:U) of material, man–hour, or use–per–hour, to their

[5]Sowa has defined the notion of *mediation* in [158] as a kind of special relation holding between entities of first order (*things*) and entities of second order (relations). Concepts being mediating belong to a third ontological level. Sowa thus proposes a special hierarchy of concepts. This hierarchy makes a distinction between concepts denoted by predicates like *marriage* which applies to instances of marriages, and concepts denoted by predicates like *married_to* which applies to pairs of individuals (persons). The discussion of whether the two predicates really denote distinct and genuine concepts is similar to the discussion in [136]. We shall use the notion of mediating tie in a different sense.

costs. We assume that such units of measure are natural numbers. A unit cost function is a function and not a map, as it may be continuous. A work index (wrkidx:WrkIdx) maps pairs of operation types and concerned objects to the cost items to which the works belong. In civil engineering the term *work* indicates that it denotes more than just the type of operation; it also denotes what object aspects are concerned.

The pragmatics behind the work index is as follows. It seems reasonable to assume that given an operation type (on:On) and a designator of an object aspect (x:X), we can determine to what cost item that work belongs. More precisely, the costs of the resources consumed by the operation concerning the object aspect, belongs to a certain cost item. We cannot, however, directly relate types of resources to cost items as the cost of the same sort of resources may belong to different cost items; e.g. costs for painting different parts of a building. Neither can we relate operation types directly to cost items as the same type of operation may consume resources belonging to different cost items, when concerning different object aspects; e.g. molding may be performed several distinct places for a building, and the costs for these works may belong to distinct cost items.

In the domain, work indices are background (sometimes almost tacit) knowledge. However, it often becomes explicit in civil engineering contracts. If we cannot express this structure, we have no way of expressing how cost frames and project plans relate. The concept thus appears to be crucial in the definition of the Galois connection.

Formally defining the mediating ties which tie together two domain concepts, adds clarity and transparency to the model which represents the relation between the two concepts.

As an extention of the model of project plans, we introduce a number of observer functions on nodes. We can observe the types of the resources consumed by the operation of the node. A specialisation of this function states, in addition, the amount (units) of each type of resource.

type
PrcIdx = Rn $\xrightarrow{m}$ UCM,
UCM = U $\xrightarrow{m}$ Cost,
WrkIdx = (On × X) $\xrightarrow{m}$ CI,
Rn,
On,
U = **Nat**

```
value
  obs_Rn_Γ: Γ → Rn-set,
  obs_Rm_Γ: Γ → (Rn →m Nat)

axiom ∀ g:Γ, rn:Rn, n:Nat •
  obs_Rn_Γ(g) ≡ dom obs_Rm_Γ(g),
```

Note, that both mediating ties go from values related to cost frames, to values related to project plans.

A work index (wrkidx:WrkIdx) can be verified according to a cost frame (cf:CF). All cost items in a cost frame should be defined in the given work index. That is:

```
value
  consistent: CF × WrkIdx → Bool
  consistent(cf,wrkidx) ≡
    frames(cf) ⊆ rng wrkidx
```

Similar, we can verify a project plan according to a work index. For a project plan, all observable operation–aspect pairs must be present in the given work index. That is:

```
value
  consistent: PP × WrkIdx → Bool
  consistent(pp,wrkidx) ≡
    (∀ g:Γ, on:On, x:X •
      g ∈ dom pp ∧ on=obs_On_Γ(g) ∧ x ∈ obs_Xs_Γ(g) ⇒
        (on,x) ∈ dom wrkidx)
```

The two functions ensure consistency between cost frames and work indices, and between project plans and work indices, respectively.

3.4 Relevant nodes and resources

We now have models of the two concepts and of the mediating ties. We define a mapping structure from so–called *relevant* nodes to *relevant resource usage*. First, we need a few definitions.

Definition 3.2 By a *resource usage* (re:Rn $\xrightarrow[m]{}$ **Nat**), we understand the resources which are consumed by an operation.

□

Definition 3.3 By a *relevant resource usage*, we understand the part of a resource usage of an operation, which concerns[6] a given object aspect.

□

A relevant resource usage can be observed:

```
value
  obs_rel_res: Γ × X → (Rn →m Nat)
```

The relevant resource usage for a node is given by adding together the resource usage for each the work concerning each object aspect involved in the operation:

```
axiom
  obs_Rm_Γ(g) ≡
    meets({rm|rm:(Rn →m Nat) •
              (∃ x:X • x ∈ obs_Xs_Γ(g) ⇒ rm=obs_rel_res(g,x))}),

  meet: (Rn →m Nat) × (Rn →m Nat) → (Rn →m Nat)
  meet(rm,rm′) ≡
    [ rn ↦ n|rn:Rn,n:Nat •
      (rn ∈ dom rm ∧ rn ∈ dom rm′ ∧ n=rm(rn)+rm′(rn)) ∨
      (rn ∈ dom rm ∧ rn ∉ dom rm′ ∧ n=rm(rn)) ∨
      (rn ∉ dom rm ∧ rn ∈ dom rm′ ∧ n=rm′(rn)) ],

  meets: (Rn →m Nat)-set → (Rn →m Nat)
  meets(rms) ≡
    if rms = {} then [ ]
    else
      let rm:(Rn →m Nat) • rm ∈ rms in
        meet(rm,meets(rms \ {rm}))
      end
    end,
```

[6] See Definition 3.1.

The same resource cannot be consumed in operations of different work, as argued in Section 3.3.

Definition 3.4 A node in a project plan is *relevant* to a given cost item, if and only if there exists an object aspect of the node such that the work given by the operation of the node and the object aspect, belongs to the cost item. This membership is defined by a given work index.

□

The set of *relevant nodes* of a project plan, given a cost item and a work index, is thus:

```
value
  rel_nds: PP × CI × WrkIdx → Γ-set
  rel_nds(pp,ci,wrkidx) ≡
    {g|g:Γ •
      g ∈ dom pp ∧
        (∃ x:X • x ∈ obs_Xs_Γ(g) ∧ ci=wrkidx(obs_On_Γ(g),x))}
```

The mapping from *relevant nodes* to *relevant resource usage* is defined as a variant of rel_nds:

```
value
  rel_map: PP × CI × WrkIdx → (Γ →m (Rn →m Nat))
  rel_map(pp,ci,wrkidx) ≡
    [ g ↦ rm|g:Γ,rm:(Rn →m Nat) •
      g ∈ rel_nds(pp,ci,wrkidx) ∧
        (∃ x:X • x ∈ obs_Xs_Γ(g) ∧ ci=wrkidx(obs_On_Γ(g),x) ∧
          rm=obs_rel_res(g,x)) ],
```

It is argued in Section 3.2.3 that the quantified x is unique.

3.5 Connecting the two concepts

The predicate which binds cost frames and project plans — our *Galois predicate* — is thus:

```
value
  φ_fp: CF × PP → (PrcIdx × WrkIdx) ~→ Bool
  φ_fp(cf,pp)(prcidx,wrkidx) ≡
    (∀ ci:CI • ci ∈ dom cf ⇒
      let (cst,cf′)=cf(ci) in
        sumcost(rel_map(pp,ci,wrkidx),prcidx) ≤ cst ∧ φ_fp(cf′)
      end)
  pre consistent(cf,wrkidx) ∧ consistent(pp,wrkidx),

  sumcost: (Γ -m→ (Rn -m→ Nat)) × PrcIdx ~→ Cost
  sumcost(gm,prcidx) ≡
    if gm = [ ] then zero
    else
      let g:Γ • g ∈ dom gm in
        sumcost′(gm(g),prcidx) + sumcost(gm \ {g},prcidx)
      end
    end,

  sumcost′: (Rn -m→ Nat) × PrcIdx ~→ Cost
  sumcost′(rm,prcidx) ≡
    if rm = [ ] then zero
    else
      let rn:Rn • rn ∈ dom rm in
        prcidx(rn)(rm(rn)) + sumcost′(rm \ {rn},prcidx)
      end
    end
  pre (∀ rn ∈ dom rm ⇒ rn ∈ dom prcidx),
```

The predicate ϕ_{fp} defines the relation between cost frames and project plans. In Formal Context Analysis [85], the relation is known as a *context*[7]. Note, that the predicate is defined using quantification over cost items (ci:CI). The reason is that usually it is the cost frame, which restricts the project plans. Project plans are made in a *constructive* way having cost frame knowledge in mind. The restrictions which project plans make on cost frames are given by the consistency predicates.

By means of the predicate ϕ_{fp}, we can define the set of *valid* project plans, as the project plans which each can be executed within a given cost frame (cf:CF); assuming the mediating ties:

```
value
```

[7] Usually meant to hold between orderings of objects and their common properties.

valid_plans: CF × PrcIdx × WrkIdx $\xrightarrow{\sim}$ PP-**infset**
valid_plans(cf,prcidx,wrkidx) ≡
{pp|pp:PP • ϕ_{fp}(cf,pp)(prcidx,wrkidx)}

Similar, we can define the set of *applicable* cost frames which are the cost frames which each apply financially, given a project plan (pp:PP); assuming the mediating ties:

value
appl_frmexps: PP × PrcIdx × WrkIdx $\xrightarrow{\sim}$ CF-**infset**
appl_frmexps(pp,prcidx,wrkidx) ≡
{cf|cf:CF • ϕ_{fp}(cf,pp)(prcidx,wrkidx)}

3.6 Galois connection and a theorem

In the following, we show that the defined connection between the two considered concepts, satisfies Galois criterion.

3.6.1 Galois connection

Let P and Q be ordered sets. Further, let p_i and q_j be elements in the sets $(P, \leq)$ and $(Q, \leq)$, respectively. From [85] we have the following definition[8] of the notion of Galois connection:

Definition 3.5 A pair of mappings (φ, ψ), with respect to the ordered sets $(P, \leq)$ and $(Q, \leq)$, is a Galois connection if φ and ψ are monotonously decreasing:

$$\forall p_i(p_1 \leq p_2 \Rightarrow \varphi p_2 \leq \varphi p_1) \tag{3.1}$$
$$\forall q_i(q_1 \leq q_2 \Rightarrow \psi q_2 \leq \psi q_1) \tag{3.2}$$
$$\forall p(p \leq \psi\varphi p) \tag{3.3}$$
$$\forall q(q \leq \varphi\psi q) \tag{3.4}$$

[8]The definition is slightly reformulated; as we explicitly express the quantifications.

The two mappings are then called **dually adjoint**.

□

We are interested especially in orderings given by set–inclusion. That is, P and Q are powersets and p_i and q_j are subsets of these powersets, respectively.

Figure 3.5–3.8 depicts the four Galois axioms (where we rename φ to $\mathcal{F}$ and ψ to $\mathcal{G}$, respectively).

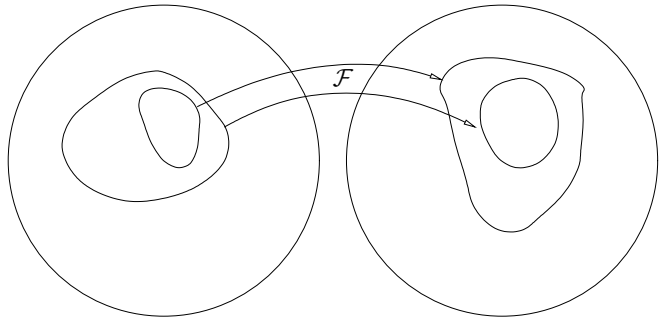


Figure 3.5: Galois connection axiom 1.

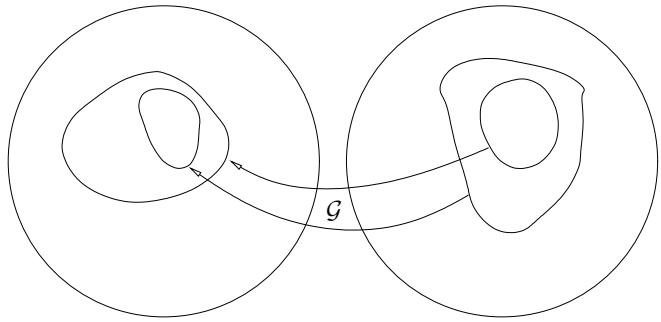


Figure 3.6: Galois connection axiom 2.

3.6.2 A theorem

We now define a mapping pair ($\mathcal{F}$,$\mathcal{G}$) between ordered sets of cost frames and project plans. The ordering is by set–inclusion. $\mathcal{F}$ maps sets of cost frames to the set of project plans which are valid within each cost frame in the given set.

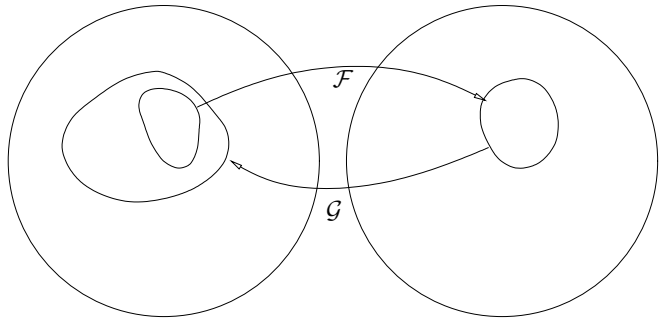


Figure 3.7: Galois connection axiom 3.

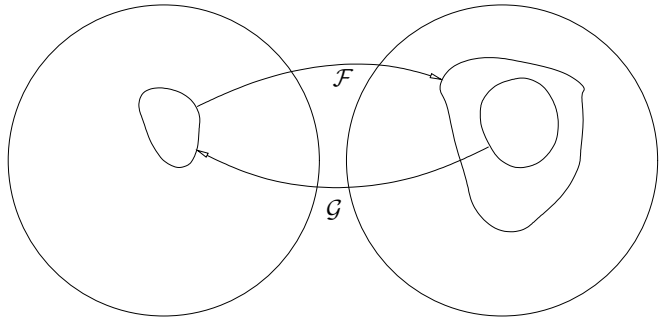


Figure 3.8: Galois connection axiom 4.

$\mathcal{G}$ maps sets of project plans to the set of cost frames which are applicable for each project plan in the given set. I.e.:

value
  $\mathcal{F}$: CF-**set** → (PrcIdx × WrkIdx) $\xrightarrow{\sim}$ PP-**infset**
  $\mathcal{F}$(cfs)(prcidx,wrkidx) ≡
    {pp|pp:PP • (∀ cf:CF • cf ∈ cfs ⇒ ϕ_{fp}(cf,pp)(prcidx,wrkidx))},

  $\mathcal{G}$: PP-**set** → (PrcIdx × WrkIdx) $\xrightarrow{\sim}$ CF-**infset**
  $\mathcal{G}$(pps)(prcidx,wrkidx) ≡
    {cf|cf:CF • (∀ pp:PP • pp ∈ pps ⇒ ϕ_{fp}(cf,pp)(prcidx,wrkidx))}

Theorem 3.6 *The mapping pair ($\mathcal{F}$,$\mathcal{G}$) is a Galois connection.*

Proof: *The proof is by inspection using Theorem 2 in [85] (first half of this theorem is repeated in the appendix of this paper). The functions $\mathcal{F}$ and $\mathcal{G}$ correspond*

to φ_R and ψ_R, respectively. The binary relation $R \subseteq M \times N$ is the binary relation between cost frame values (cf:CF) and project plan values (pp:PP). The binary relation is defined by the predicate ϕ_{fp}; assuming $\forall$–quantification over the mediating ties (prcidx:PrcIdx) and (wrkidx:WrkIdx). X and Y correspond to finite sets of values having type CF and type PP, respectively; x and y are values in these sets.

□

The functions $\mathcal{F}$ and $\mathcal{G}$ are order–reversing. The connection is thus incrementally decreasing as shown on Figure 3.5–3.8.

In the following we analyse the meaning of the connection looking at what knowledge the two domain concepts represent.

3.7 Analysis of the connection

What is first recognized in Theorem 3.6 is that neither the definition nor the content of the predicate ϕ_{fp} have influence on whether the mapping pair satisfies Galois criterion. E.g. reversing the order of the *sumcost* comparison in the predicate would not have changed this. However, it would have changed the *extension* of the predicate. Instead of applying to project plans which can be executed within cost frames, the predicate would apply to project plans which cannot be executed within cost frames. The result of this paper is thus more than showing that Galois criterion is indeed satisfied. It also illuminates the meaning of the extension of the predicate. In this sense, the notion of Galois connection is a framework which forces us to be explicit about a denotational relation between concepts and between models. Defining a relation explicitly adds clarity and trasparency to the claimed connection.

The essence of what it means for two domain concepts to satisfy this criterion is to be found in the extension of the predicate; given the same mediating ties. In the following we explain what each Galois axiom means in the context of cost frame and project plans.

- each set of cost frames maps to a set of project plans; namely the project plans that each can be executed within each cost frame. Let us call this map $\mathcal{F}$.
- each set of project plans maps to a set of cost frames; namely the cost

frames within which each of the project plans can be executed. Let us call this map $\mathcal{G}$.

3.1 for all sets of cost frames, if one set p_1 is a subset of another p_2, then the set of project plans $\mathcal{F}(p_2)$ should be a subset of the set $\mathcal{G}(p_1)$.

3.2 for all sets of project plans, if one set q_1 is a subset of another q_2, then the set of cost frames $\mathcal{G}(q_2)$ should be a subset of the set $\mathcal{G}(q_1)$.

3.3 for all sets of cost frames p, the set is a subset of $\mathcal{G}(\mathcal{F}(p))$.

3.4 for all sets of project plans q, the set is a subset of $\mathcal{F}(\mathcal{G}(q))$.

For the two functions $\mathcal{F}$ and $\mathcal{G}$ to be a Galois connection, shows that there is truly a denotational relation between the two concepts considered. This basically means, that the the relation has strong mathematical ordering properties, that it is sound and consistent, and that it is based on set–theory.

However, the most important result is that the relation satisfying Galois criterion complies with an ordering intuition. This intuition is that if we raise cost amounts, more project plans are executable, and breaking down cost items, makes less project plans executable. The essence here is that there is a concept of classification which defines the duality between the ordered sets falling under the two concepts respectively.

3.8 Conclusion

We have presented models of two civil engineering concepts: cost frames and projects plans. We have shown that these two concepts, which can be modelled and analysed in isolation, can be related by introducing what we have called mediating ties. The relation between the two concepts considered satisfies Galois criterion. Thereby, we have the possibility of checking whether a project plan can be executed within the financial restrictions of a cost frame, and vice versa. We thus believe to have captured an essential concern of civil engineers in practice, and — by the mathematical strength of the models and the principle — to have contributed to a foundation for advanced planning tools in civil engineering.

Relating domain concepts by means of a Galois principle, we believe is applicable for a large set of civil engineering domain concepts. Thus, we may wish to relate concepts like conceptual designs and requirements, city plans and location

plans, etc. Thereby, we apply a Galois principle to built an ontology of civil engineering concepts. Such an ontology is, contrary to many other civil engineering ontologies, founded on a mathematical relation between concepts which again is founded on set–theory. The concepts considered thus relate mathematically, and not by convention.

Appendix

The following is a reprint of a theorem from [85]:

Theorem 2. *For every binary relation $R \subseteq M \times N$, a Galois connection (φ_R, ψ_R) between M and N is defined by*

$$\varphi_R X := X^R \ (= y \in N | xRy \ for\ all\ x \in X)$$
$$\varphi_R Y := Y^R \ (= x \in M | xRy \ for\ all\ y \in Y).$$

$\diamond$

Part III

Design

Chapter 4

From rough to final designs by incremental set–inclusion of properties

Prof. Anders Ekholm, Lund Institute of Technology, is co–author of this paper, which appeared in Turk Z. and Scherer R. (eds.) eWork and eBusiness in Architecture, Engineering and Construction. Swets & Zeitlinger Publishers.

Abstract: Design of buildings is a complex task in which ideas are sketched and communicated, by representations that are incrementally elaborated from the early rough sketches to the final design. We claim, that today's model–based design tools are restricted from fully supporting this process as they are founded on the principle that objects are instances of static types. Such systems do not offer work with objects being incrementally specialised according to their properties, and neither do they offer dynamics of the underlying type system.

The present paper elaborates on a property–oriented approach as a foundation for design tools facilitating incremental design based on set–inclusion of properties. We emphasize the formal foundation for incorporating such dynamics, and we specify requirements for tools facilitating incremental design and offering improved semantic support.

4.1 Introduction

The introduction of computer–aided design (CAD) using model–based tools has revolutionized many industries such as the automobile industry, the computer hardware industry, and the building industry. However, in building design, the notion of creativity has suffered in favour of efficiency. One reason may be that most model–based design tools do not support incremental design; i.e. a process in which objects of the current model are being incrementally specialised according to their kinds.

In civil engineering and architecture, design is a creative process in which ideas are sketched and communicated, by representations that are incrementally elaborated from early rough sketches to the final design. In order for a model–based design tool to support (or at least not obstruct) creativity in this way, it should facilitate dynamics on four levels: (i) on the parts of the building, (ii) on the properties of parts, (iii) on other binary relations between parts, and (iv) on the underlying type system. Most design tools facilitate the first, whereas they lack of functionality facilitating the three succeeding.

In this paper, we elaborate on a property–oriented approach as a foundation for design tools facilitating incremental specialisation of designs, based on set–inclusion of properties. We do so by offering a framework for dynamic management of objects and properties. This framework is well–founded in mathematical and computer science theories and disciplines like Formal Concept Analysis, Lattices, and Galois connections. Furthermore, it strongly relates to formal methods and domain modelling as research disciplines of computer science [19, 15]. We sketch requirements for design tools facilitating incremental design and offering improved semantic support.

4.1.1 The conceptual level

Traditionally, CAD tools were made to create artefact descriptions like drawings, and for speeding up the design process by ensuring consistency, facilitating reuse, etc. With the introduction of 3D modelling and visualisation, graphical presentations like drawings (in a wide but still syntactical sense), are more important than ever.

However, there is more to a design process than the graphical entities of some presentation. Such a presentation can be seen as piece of syntax (i.e. a meaningful sequence of symbols) that certainly stands for something and is made according to some idea of an artefact. Cognitively, we understand artefact de-

scriptions like drawings as *presentations* of some conceptual model. In computer science, we would say that artefact descriptions are computerized interpretations or evaluations of the conceptual design model. Such a model is present in all stages of any design process and necessarily precedes any presentation-aimed syntax.

In order to develop design tools, we need to establish a conceptual understanding of the process of designing and of design ideas.

4.1.2 Survey

Design as an incremental process is not a new concept. As an opposition to design tools focusing on the late stages of design, the Swedish BAS•CAAD project was initiated [64, 67, 75, 167], suggesting a computerized tool supporting design in the early stages. Also, the notion of *schema evolution* has been emphasized in context of design. For example, Hakim and Garrett propose an *Object–Centered* approach to modelling as a contrast to *Class–Centered* modelling, aiming the same as we, but using an informal reference frame [97, 87].

4.1.3 Suggested framework

In this paper, we elaborate on the *property–oriented* approach presented in the BAS•CAAD project. We see design as a process of exploration, which includes choosing among alternatives, and incrementally adding parts, properties of parts, and other relations between parts. In the work mentioned in this paper, we focus the discussion on incremental property determination. What separates it from other work on incremental design is, that we intend to put on a logico–computer science perspective. Thereby, we go back and investigate the formal foundations for CAD. Especially, we focus on the notion of multiple inheritance of properties as a fundamental issue in design.

We claim that the (apparently) natural process of going from rough sketches to more and more precise designs is not entirely supported by today's commercial CAD tools. Rather, such tools emphasize the facility of incrementally adding objects (parts) of pre–defined static types, thereby focussing on partonomic (*part–whole*) relations [7] and not on taxonomic (*kind–of*) relations [94]. The latter sort, we claim is as important to design as the former.

We agree with Turk et al [169], that static type structures in the object–oriented paradigm restrict the practitioner and imply some sort of blindness. Though,

we seek to approach the problem without issuing it as for or against *hermeneutic constructivism*. In our approach, we could say that incrementally stating the properties of objects in a design model, makes it possible for the designer to reflect on the design. Actually, making local, temporary restrictions like determining properties and even constraints, certainly offer a foundation for deciding how the design should be — or should not be. Total blindness in a design process may appear if we along the way keep too many open ends — we may not get anything done at all.

The framework, suggested here, sees properties as the primary ontological entities being referred to in descriptions of artefacts. Concerning representation, we consider properties to exist on a dynamic run–time level rather than on a static. Thereby, we obtain the dynamics mentioned above.

We rely on formal theories like Formal Concept Analysis, Lattices, and Galois connections from knowledge engineering [85, 96, 95] to establish this framework. Furthermore, we use The RAISE Specification Language (RSL) [134, 135] to specify requirements for design tools facilitating incremental design and offering improved semantic support, thus being suitable for distributed design. The formulae in RAISE are mathematically precise descriptions of the presented ideas, and aim to add a computer science perspective. However, the formulae are supplementary in the sense that they are not crucial for the understanding of this paper. For simplicity, formal specification of so–called well–formedness has be omitted.

4.2 The design process

Design may be seen as a problem solving process similar to problem solving in everyday life or in science. Such a process starts by expressing a problem definition. In order to recognize a problem one usually need some sort of goal, but not necessarily knowledge of how to reach that goal. Thus, the notion of a problem can be defined as *lack of solution knowledge in relation to background knowledge and goal* [40]. A goal can be defined as an intended state of a system; i.e. the properties of the system at a certain time. In design, a goal could be understood as a satisfactory behaviour of an artefact.

Stating the problem definition is followed first by synthesis, describing a hypothesis or tentative problem solution (technical solution), and then by analysis which investigates the proposed solution. The synthesis question is: "*Which system has these properties*?" The analysis question is the inverse: "*What properties does this system have*?" Synthesis may be regarded as starting from a

functional view on the system, while analysis starts from a compositional view. The result of the analysis is added to the background knowledge and may lead to a revision of the goal. During the design process, hypotheses and tests are made alternately, and the properties of the intended artifact are determined incrementally. The design cycle — which by Simon is called the "*Generator-Test Cycle*" [154], proceeds until a satisfactory solution has been reached. The problem solving process is illustrated in Figure 4.1.

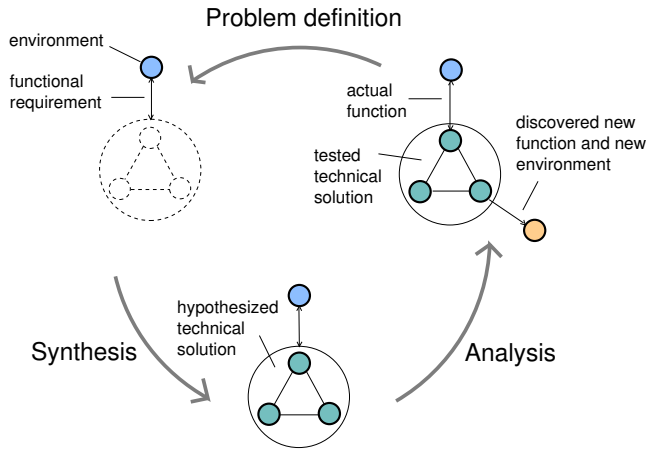


Figure 4.1: The problem solving process [65].

4.2.1 Incremental design: An example

Consider the process of designing a house to be built. Usually, the overall structure is sketched before large details like doors, windows, the sort of walls, etc. can be specified; although that may not always be the case.

Figure 4.2 shows three stages of such a design: (a) a drawing of rough lines indicating the exterior structure, (b) a drawing of more precise geometry, doors added and wall type information stated, and (c) a drawing which in addition states the widths and lengths of walls.

We thus go from a rough sketch to more and more detailed specifications. By a rough sketch, we understand an artefact description which is *less* precise than a (relatively) more detailed specification.

Using predicate logic, we could formalize aspects of a wall object at two of the stages respectively as:

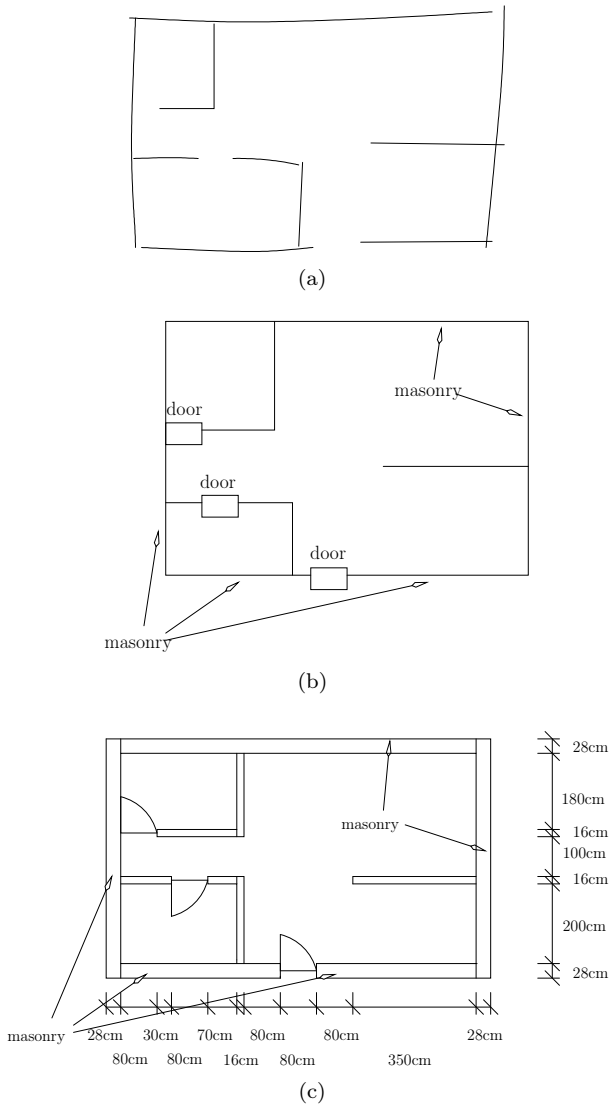


Figure 4.2: Rough sketch (a), simple drawing (b), more detailed drawing (c).

$$\mathsf{wall}(x_1) \wedge \mathsf{straight}(x_1) \wedge \mathsf{masonry}(x_1) \tag{4.1}$$

$$\mathsf{wall}(x_1) \wedge \mathsf{straight}(x_1) \wedge \mathsf{masonry}(x_1) \wedge \mathsf{width}_{28}(x_1) \wedge \mathsf{length}_{568}(x_1) \tag{4.2}$$

We see, that Formula 4.1 and 4.2 are bound by implication from the latter to the former. That is, the former is a more rough description of the object x_1 than the latter.

4.2.2 Many–sorted knowledge and multiple inheritance

An essential issue of design is that of adding different, many–sorted knowledge to an artefact description. That is, designing a building means adding different *incomparable* sorts of knowledge like material, colour, texture, etc. Furthermore, designing is also the process of putting together parts on the conceptual level, thereby forming wholes, as well as adding other binary relations between objects.

Seen this way, design is: (i) multiple inheritance as set–inclusion of properties like being red or made of wood, (ii) set–inclusion of parts like being composed by a number of bricks, and (iii) set–inclusion of other binary relations like for two walls to be parallel and two meters appart.

Figure 4.3 shows how multiple inheritance of properties is meet ($\times$) and design moves in separate incomparable directions origins from join ($+$) in a lattice structure [13].

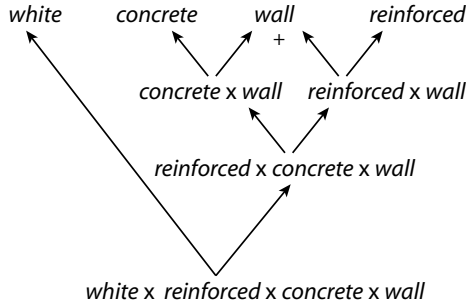


Figure 4.3: Multiple inheritance.

4.3 Tools for design

In context of tools for design, the introduction of the object–oriented paradigm has *not* been more revolutionary than adding some convenient ways of working with data. That is, the real difference between classes and traditional data types is merely on some technicalities. In essence, classes are (as types) still static.

The following two sections state the importance of abandoning this restriction by indicating requirements for design tools for incremental design.

4.3.1 Dynamics on properties

According to the object–oriented paradigm, objects are created as belonging to a certain class.

This has at least two consequences: (i) we cannot speak of loosely defined objects being incrementally specialised during run–time of the design tool, and (ii) objects cannot shift classes.

These facts, we claim, delimit today's design tools from fully supporting the creative process of design.

4.3.2 Dynamics in type system

A class in the object–oriented paradigm is a type in the language in which the tool is written. Thus adding a class definition or an attribute denoting a property, usually requires recompilation of the program code. This implies that classes and attributes cannot be changed or added after instantiation of any objects [97, 87].

In order to facilitate dynamics according to properties of objects and of the underlying type system, we need to move properties and relations of objects from the static type level to the dynamic run–time level [67]. We do so by establishing a generic system in which properties and relations are values and thus can be managed at run–time. Furthermore, we have objects, which are also values. These map to their properties in an object environment store. The latter, we shall call an *artefact model*.

This solution, however, has implications as we shall see in Section 4.6.

4.4 Artefact model

By an artefact model, we understand a configuration of objects, i.e. maps from objects to their properties and parts, respectively; and a map from conceptual

object pairs to relation designators. We found the specification of artefact models on the notion of *design moves* such that artefact models indeed facilitate incremental design.

4.4.1 Design move

The three stages of design from Section 4.2.1 are related by design moves. A design move is a transition from one artefact model a to another artefact model b; denoted $\Phi(a) \equiv b$. A design move $\Phi(a) \equiv b$ is valid, iff b is a description of an artefact idea which is a *relevant successor* to the artefact idea corresponding to the description b [79]. Logically, the description b is a relevant successor of a, if and only if the artefact described taxonomically as well as partonomically and according to other binary relations, is a specialisation of the artefact described by a.

4.4.2 Specification

A formal specification of an artefact model should accomplish incremental specialisation and we thus specify the notion as a record with three entries: (i) The taxonomical relations as a map (t–map) from objects (x:X) to sets of properties (p:P), (ii) the partonomic relations as a map (p–map) from objects to their object parts, and (iii) additional binary relations as a map (r–map) from pairs of objects to sets of relation designators (υ:Υ):

type
Θ :: t:X $\xrightarrow{m}$ P-**set** p:X $\xrightarrow{m}$ X-**set** r:(X × X) $\xrightarrow{m}$ Υ-**set**,
X, P, Υ

Note, that part-whole relations are singled out as a special case of binary relations between objects.

The wall modelled in Formula 4.2, can then be represented by the following artefact model:

value
x_1: X, wall, straight, masonry, $width_{28}$, $length_{568}$:P,
θ:Θ = mk_Θ([x_1↦{wall, straight, masonry, $width_{28}$, $length_{568}$}],
[x_1↦{}],[ ])

Using the specification of Θ, we define the notion of design move as:

```
value
  Φ: Θ → Θ,
  ≤: Θ → Θ → Bool

axiom ∀ θ, θ′:Θ •
  θ ≤ θ′ ≡
   let (t,p,r)=θ, (t′,p′,r′)=θ′ in
      (∀ x:X • x ∈ dom t ⇒ x ∈ dom t′ ∧ t(x) ⊆ t′(x)) ∧
      (∀ x:X • x ∈ dom p ⇒ x ∈ dom p′ ∧ p(x) ⊆ p′(x)) ∧
      (∀ (x,x′):X×X • (x,x′) ∈ dom r ⇒ (x,x′) ∈ dom r′ ∧
         r(x,x′) ⊆ r′(x,x′))
   end,
  Φ(θ) as θ′
  post θ ≤ θ′
```

It can be shown that the relation ≤ defines a partial ordering on artefact models (see Chapter 5).

4.5 Towards property–orientation

The backbone of any ontology–based information system is the *kind–of* relation and the relation between objects and their properties. The foundation for these relations is the mathematical notion of a *Galois connection* which can be said to hold between two power sets [85]. A Galois connection between sets of objects and sets of their common properties has the convenience that it defines a partial ordering of the properties. Furthermore, the partial ordering is a lattice which facilitates advanced and flexible querying and knowledge management.

In the following sections, we give a short introduction to Galois connections in order to justify the presented specification of artefact models and its focus on properties.

4.5.1 Galois connection

We can say that an object x has the property y. For that, we write xRy where R is called the *incidence relation* [85]. The correspondence between objects and

their common properties becomes quite an important relation, as the functions giving the common properties of an object set and the residual function appears to be a Galois connection.

Definition 4.1 A pair of mappings (f, g), with respect to the power sets X-**set** and P-**set**, is a Galois connection iff f and g are monotonously decreasing:

```
type
  X, P

value
  f: X-set → P-set,
  g: P-set → X-set

axiom
  (∀ xs,xs′:X-set • xs ⊆ xs′ ⇒ f(xs′) ⊆ f(xs)) ∧
  (∀ ps,ps′:P-set • ps ⊆ ps′ ⇒ g(ps′) ⊆ g(ps)) ∧
  (∀ xs:X-set • xs ⊆ g(f(xs))) ∧
  (∀ ps:P-set • ps ⊆ f(g(ps)))
```

The two mappings are then called *dually adjoint* [85]. The above rather sloppy specification assumes the existence of only *one* incidence relation.

We now define f and g, as we model the incidence relation as a map (m:M) from object–property pairs to boolean values:

```
type
  M = (X × P) m→ Bool

axiom ∀ x:X,p:P,xs:X-set, ps:P-set, m:M •
  f_m(xs) ≡ {p|p:P • (∀ x:X • x ∈ xs ∧ (x,p) ∈ dom m ⇒ m(x,p))},
  g_m(ps) ≡ {x|x:X • (∀ p:P • p ∈ ps ∧ (x,p) ∈ dom m ⇒ m(x,p))}
```

Note, that we for f and g assume the presence of a formal context (m:M). The pair (f, g) satisfies the criteria for being a Galois connection [85]. This has the advantage that properties are partially ordered and forms a lattice structure, which can be utilized in semantic queries and support facilities in design tools as sketched in Section 4.6.2.

The important taxonomical relation is modelled in Θ as the t–map. The relation is essential; a backbone in all ontological systems, and can be formulated as unary predicates, like being a wall ($\mathsf{wall}(x)$), or having a length of $4cm$ ($\mathsf{width\_4}(x)$). The partial ordering of properties is fundamental, as it e.g. places the property of being a door as a more general property than e.g. being a left–hand door, in the lattice.

The t–map, however, is not the only structure forming a lattice. Also the p–map and r–map form lattices because of their set–based nature.

In knowledge engineering, attempts have been made for unifying taxonomic and partonomic relations using the Pierce product as notation for attribution; [34]. In Θ, the three maps can be seen as three different aspects of the artefact described [155].

4.5.2 Relation to the BAS•CAAD ThingClass

In this section, we relate the presented artefact model to the *ThingClass* of the BAS•CAAD project [68], from which it was inspired. A ThingClass is defined as a 6–tuple of attribute sets, denoted: $T = (T_G, T_C, R_I, T_E, R_E, A_U)$. T_G is the set of generic or superclass attributes, T_C is the set of composition attributes, R_I is the set of internal relations, T_E is the set of environment attributes, R_E is the set of external relations, and A_U is the set of unary attributes which represent intrinsic properties of systems.

The specification of the presented artefact model Θ can be considered a projection of the above ThingClass specification.

The members in the union set of generic or superclass attributes (T_G) and unary attributes representing intrinsic properties (A_U) each designate properties (p:P) in Θ. We have not made a distinction between class attributes and intrinsic properties, as we formally cannot make this distinction. Both may be denoted by unary predicates, so the distinction is merely intentional.

The set of composition attributes, corresponds to the p–map in Θ.

The sets of environment attributes and external relations have been omitted in Θ. In the ThingClass these aim at modelling functional requirements (or behaviour) of an artefact, but such information might exist on a different level of description. Formally, instantiating a binary relation between an artefact and the environment implies considering that environment piece as part of the artefact. Rather, it is combinations of properties and relations that makes

the artefact functional. E.g. a certain combination of dimension and material makes a wall resist fire. Thus, functional requirements are to be modelled as a mathematical function ranging over sets of properties and relations. This way we are able to model what Bunge's calls *dispositional properties* [38].

A further study of this area could be founded on understanding properties as functions of events [153].

4.6 Design tool requirements

In this section, we specify some basic requirements for design tools facilitating incremental design and improved semantic support. We found this specification on an organisation of information levels. We have already mentioned the conceptual level on which artefact descriptions belong. In addition, there are two more levels: The *presentation* level on which visualisations of artefact models belong, and the *Conceptual level!semantic* level on which the semantical functions of properties, relations, design moves, etc. belong. The mathematical function modelling dispositional properties (see Section 4.5.2) appears as a special semantic function (see [61] for another denotational semantic treatment of the domain of civil engineering).

Information on the presentation level is the result of computerized interpretation of information on the conceptual level, according to information on the semantic descriptive level.

Figure 4.4 shows the semantic relations between the three levels.

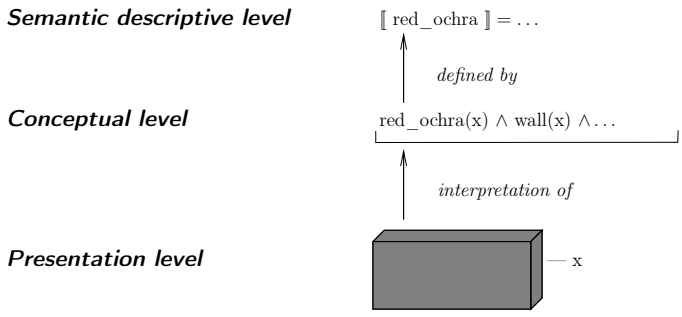


Figure 4.4: Information Levels.

Adding properties and various sorts of relations to an artefact model, we consider

a special case of semantic descriptive information (although it certainly differs from the understanding of specifying the semantics of properties).

4.6.1 Semantics of design moves

The semantics of design moves can be described as functions from one artefact model to another. That is, the various sorts of moves like adding a part, ascribing a property to an object, etc. are considered syntactical commands (c:Cmd) on the semantic descriptive level. The semantics (specified using the semantical parentheses ⟦ ⟧) we model as the result/effect of performing the move; here using a denotational approach [61]:

type
  Cmd == addobj | addprop(p:P) | addpartrel(x:X, x′:X) |
    addrel(z:X, z′:X, υ:Υ),

value
  ⟦ ⟧: Cmd $\xrightarrow{\sim}$ Θ $\xrightarrow{\sim}$ Θ,
  new: Θ → X

axiom ∀ c:Cmd, θ, θ':Θ, p:P, x,x′:X •
  ⟦ addobj ⟧(θ) ≡
    let x=new θ **in**
      mk_(t(θ)†[x ↦ {}],p(θ)†[x ↦ {}],r(θ))
    end,

  ⟦ addprop(p) ⟧(θ) ≡
    mk_Θ(t(θ)†[x ↦ t(θ) ∪ {p}],p(θ),r(θ)),

  ⟦ addpartrel(x, x') ⟧(θ) ≡
    mk_Θ(t(θ),p(θ)†[x′ ↦ p(θ) ∪ {x}],r(θ)),

  ⟦ addrel(x, x', υ) ⟧(θ) ≡
    mk_Θ(t(θ),p(θ),r(θ)†[(x,x′) ↦ r(θ) ∪ {υ}]),

4.6.2 Semantic support

The presented property–oriented framework facilitates a large number of semantic applications. Of these, we emphasize: Querying design models (e.g. as

conformance checking), and merging artefact models. From an artefact model, we can extract the formal context (m:M) by determining the property–set (ps:P-**set**) present in the artefact model. We then build the map for the formal context by pairing objects (x:X) from the artefact model with each property in *ps* and mapping the pair to a boolean value (b:**Bool**) stating whether or not the object has this property. This context serves as the taxonomical information we need in simple querying.

value
  X_M: Θ → M
  X_M(θ) ≡
    let ps={p|p:P • (∃ x:X • x ∈ **dom** t(θ) ∧
      p ∈ t(θ)(x))} **in**
      [(x′,p′)↦b | x′:X,p′:P,b:**Bool** • x′ ∈ **dom** t(θ) ∧
      p ∈ ps ∧ b = p ∈ t(θ)(x)]
    end

A simple form for query is one which returns the set of objects that satisfies certain criteria according to possessed properties and relations. That is, a query (q:Q) is a tuple of the form (P-**set** × (ϒ $\overrightarrow{m}$ X-**set**)). Leaving all X-**set** empty yields partially evaluation assuming some ϒ–relation to *any* object.

type
  Q = P-**set** × (ϒ $\overrightarrow{m}$ X-**set**),

value
  $\mathcal{I}$: Q → Θ → X-**set**
  $\mathcal{I}$(ps,rel)(θ) ≡
    {x|x:X • (∀ p:P • p ∈ ps ⇒ p ∈ t(θ)(x)) ∧
      ∀ υ:ϒ • υ ∈ **dom** rel ⇒
      υ ∈ **dom** r(θ) ∧
      (∃ x′:X • x′ ∈ rel(υ) ⇒ υ ∈ r(θ)(x,x′))}

The above can be specialised such that interpretations of queries return sub–models of artefacts. This can be convenient when writing various extraction applications. Furthermore, by specifying requirements as sets of properties and relations, it is possible to perform conformance checking on artefact models.

As a result of distributed design processes, e.g. web–based, a function for merging two artefact models may be useful. The function should, given two artefact

models, state what possible inconsistency there is. Such information is given in terms of artefact models; i.e., the set of objects mapping to conflicting properties, and the objects making a cycle in the part–whole relations, see [134] for a formal definition. We assume, the existence of a predicate *conflict* stating whether or not two properties are mutually exclusive; based on some "universal" taxonomy represented as a formal context.

value
  merge: $\Theta \times \Theta \to M \to \Theta$
  merge(θ,θ')(m) **as** θ''
  post ($\forall$ x:X • x $\in$ **dom** t(θ'') $\Rightarrow$
    ($\forall$ (p,p′):(P×P) • {p,p′} $\subseteq$ t(θ'')(x) $\Rightarrow$ conflict(p,p′)(m)) $\wedge$
      p(θ'') = [x↦xs|x:X,xs:X-**set** •
      (x $\in$ p(θ) $\wedge$ x $\in$ p(θ') $\Rightarrow$
        xs=p(θ)(x) $\cup$ p(θ')(x)) $\vee$
        (x $\in$ p(θ) $\Rightarrow$ xs=p(θ)(x)) $\vee$
          (x $\in$ p(θ') $\Rightarrow$ xs=p(θ')(x))] $\wedge$
  is_cyclic(p(θ''))),
  conflict: (P × P) $\to$ M $\to$ **Bool**,
  is_cyclic: (X $\xrightarrow[m]{}$ X-**set**) $\to$ **Bool**

Simple consistency of *one* artefact model (θ:Θ) can be obtained by using the empty artefact model as second argument. That is, merge(θ,([],[],[])) applied on some formal context, e.g. X_M(θ).

4.6.3 Accumulating design knowledge

Even though we have assumed that an artefact model only applies to one artefact idea, the notion has more potential. Consider the case of making a whole series of artefact models $\theta_1, \theta_2, \ldots, \theta_n$. These, we assume represent different artefact ideas, each on their "final" stages of design. The knowledge within these might be important for future design processes in two ways: (i) Each artefact model defines taxonomic relations, and (ii) each artefact model can be seen as examples of how to model certain artefacts. The two can be quite important in knowledge engineering of artefacts. It can be shown that formal contexts representing taxonomic relations easily can be added incrementally. Thereby, knowledge of *kind–of* relations can be built up and concepts involving various incomparable properties can be deduced [85]. Such concepts can then be made predefined types such that efficiency of design (like we have it in today's commercial CAD tools) is achieved. Furthermore, collecting previous artefact models together

with the semantic definitions of e.g. properties, we have a (though primitive) foundation for defining concepts by their extentions. That is, simply by visualising previous objects falling under the concept in question.

4.7 Conclusion: Beyond drawings

We have presented a property–oriented framework for design tools supporting incremental design. In order to do so, we have specified the notion of an artefact model storing taxonomic and partonomic information, as well as information about other binary relations of the artefacts. An achievement is that properties and relations can be added incrementally to the current artefact model. That is, we abandon the idea of static type systems and move properties and relations to the dynamic run–time level. Furthermore, we have indicated how new properties and relations can be introduced by formally specifying their semantics. We have argued that the aspects of any artefact model separately satisfies Galois criterion, and thus makes the specification a solid foundation for tools offering improved semantic support. In this context, we have specified some basic requirements.

However, the framework can be taken further. In this paper, we have not really relied on artefact descriptions as syntactical documents like drawings. Rather, we have focused on the underlying conceptual framework, being the semantical one. A similar framework could certainly be defined for other sorts of civil engineering tools like for managing construction specifications or handling contracts. We believe that a wide range of civil engineering tools might benefit from a semantic treatment and thus can be founded on a property–oriented framework.

Chapter 5

Incremental building design as lattices

Abstract: The paper explores the formal and philosophical foundation for incremental design by introducing the notion of artefact models and an ordering relation between such models. As a step towards advanced design tools, we suggest that the various design stages are recorded in a structural way which complies with a mathematically defined ordering relation between the stages. Two mathematically well–founded approaches are explored: An approach based on class partition, and an approach based on lattices. We argue that the latter is most suitable as it facilitates a distinction between two sorts of design moves: Those which explore design aspects, and those which compose partial solutions. The former kind of moves origin from join in the lattice. The moves include adding objects, properties of objects, and relations between objects. The latter kind corresponds to lattice meet. The lattices for recording design processes are called *design lattices*. In these, nodes correspond to artefact models and edges correspond to design moves. We show that the design moves define a partial ordering of artefact models, and that this ordering satisfy lattice criteria.

5.1 Introduction

Consider the process of designing a steel profile for a bridge structure. Through the process, a number of properties are selected. These could be: a length of $2400mm$, a height of $28mm$, a width of $44mm$, the property of being made of steel, and having an H–shaped profile. Figure 5.1 depicts such a steel profile.

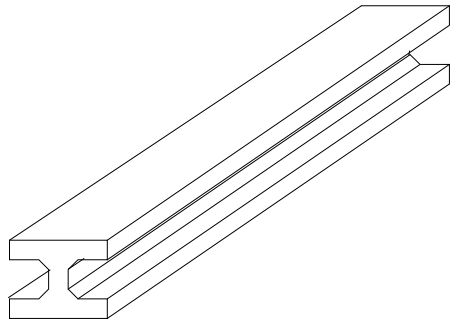


Figure 5.1: A steel profile.

The properties mentioned may be conceived and selected for the design in any order without resulting in different object conceptions. The reason is that the property domains of length, height, width, material, etc. are *incomparable sorts of design knowledge*. A property domain, following Gärdenfors, is an ontological space of properties similar in kind [86]. E.g. colour properties belong to the same property domain. Properties belonging to the same domain exclude each other. E.g. an object cannot have two distinct masses. Contrary, properties from incomparable domains can co–exist for an object.

Properties from incomparable domains are brought together in order for the object in mind to have certain functionality. In general, we say that the object has certain *causal dispositions*. This means that the object can react in a certain way given certain stimuli or when put in certain situations. We thereby follow philosophical doctrines like Shoemaker's [153] in which properties are related to the notion of causation; the theory of causes and effects. For example, the mentioned dimension and material properties may give the profile the strength to resist a certain pressure on the middle. This strength is a causal disposition of the beam. The problem of selecting a set of properties such that an object will have certain functionality is called a *design problem* [45].

Bringing together a number of properties from incomparable domains is in knowledge engineering and formal ontology known as *multiple inheritance*. We believe that this concept is crucial to that of designing.

As a step towards advanced and sophisticated planning and design tools, we wish to record design stages and the relation between them; the design moves [149]. The systematic design practitioner may strive to be aware of the design moves being made. It is background and sometimes tacit knowledge whether one design is more detailed and specialised than another, whether two alternative design solutions are in conflict or can be combined, etc.

In this paper, we introduce a structure and approach for recording and structuring design processes. Such a structure is to maintain an ordering relation between design stages. Order–theoretically, we can take three seemingly different approaches.

The first approach partitions a class of objects falling under some general design concept like hospital or bridge. The partition process is performed recursively until a satisfactory design concept is reached. We call this approach the *class partition approach*. The approach complies with the mathematical notion of class partition of individuals and thereby seems attractive.

The second approach origins in an empty design to which objects, properties of objects, and relations between objects, are added. Different design stages can be combined by means of multiple inheritance. The approach is called *design lattices*, as it is based on the mathematical notion of lattices which makes the approach attractive.

The third approach origins in a notion of *mereological*[1] universe which is the sum of all objects. This sum is partitioned recursively until a satisfactory object sum is reached[2]. Objects are, however, not specialised by their kinds, which makes the approach unattractive for design processes going from rough sketches to final designs[3].

The notion of design is complex. It involves such notions as knowledge and believe, causal dispositions, natural laws, creativity, aesthetics, non–deterministic and even irrational decisions. Also, a given design problem can be approached and solved in various ways. Selecting a property in one domain may restrict the space of properties from other domains, necessary for solving the design problem at hand. E.g. designing the H–profile to be of weaker material requires it to be larger in cross section, in order for it to have the same strength. The restrictions are defined by ontological bindings; most important the natural laws of physics.

[1] We shall not discuss the notion of mereology in this paper, but refer to [155].

[2] In fact the reverse process could also be considered in which case the process (besides the process of specialising objects) has similarity with design lattices.

[3] This subject has been treated in Chapter 4.

The aim of introducing design lattices is, however, deliberately terse: We do not explain *how* nor under *what mental conditions* the designer conceives design ideas or makes choices. Neither, do we attempt to order design choices according to ontological bindings.
Our aim is to record — not to restrict — the design process. We believe that design lattices, in its order–theoretics is a suitable approach and makes a good formal foundation for conceptual design tools aimed for all stages of design.

Design lattices facilitate browsing and combining of design configuration. By browsing we understand reviewing previous design configurations. Combining design configurations is an important ingredient in todays distributed world.

In Section 5.2 we present a perspective on the design process and define the notion of design move based on a philosophical and scientific understanding of *problem* and *solution*. We argue in Section 5.3 that the seemingly attractive approach of class partition falls short of capturing important design process issues. We claim that the reason is that the approach lacks a distinction between two sorts of design moves: design moves *by aspect* and design moves *by configuration*. The notion of design lattice is then introduced in Section 5.4 in order to facilitate such a distinction which incorporates multiple inheritance.

In Section 5.6 we present a formal model of building design configurations. Further in Section 5.7, we define a relation between such configurations and we show that the relation is a partial ordering satisfying basic lattice axioms.

Throughout the paper, we use the RAISE Specification Language [134, 135] to specify mathematical precise formulations of the presented ideas.

5.2 The design process

We found our work on a design understanding called *incremental design*. It is a design conception which sees design as an exploration process in design which knowledge is incrementally added to design configurations. More precisely:

Definition 5.1 (Incremental design) Incremental design is a process in which objects, properties of objects and relations between objects are designated and alternately added to a design configuration.

Definition 5.2 (Design configuration) By a *design configuration* we understand a conceptual design representation.

When a design configuration represents the design of an artefact, we speak of an *artefact model*.

Definition 5.3 (Artefact) By an *artefact*, we understand a physical object which is the product of human actions and which has been intentionally produced for some purpose [100].

Incremental design has been explored intensively in the BAS•CAAD project [64, 67, 75, 167] which focused on design at the early stages. Formal treatments of incremental design include [62] (Chapter 4) which founded ideas for the present paper.

5.2.1 Problem and solution

The notion of incremental design is founded on a philosophical and a scientific understanding of the notions of *problem* and *solution*. According to Bunge a problem can be seen roughly as lack of solution knowledge [40]:

> "A problem is a knowledge gap, and a problem solving process is one aiming at filling such gap..."

The definition is a general one; although, there are sorts of problems which do not match it, e.g. computational problems which can be described but not solved. However, the definition seems attractive when considering the notion of design.

Through an iterative process, knowledge for a solution is collected into a tentative solution. In each iteration, the tentative solution is validated against the problem at hand. The result shows whether the solution has been reached, alternatives should be considered, or whether more design knowledge needs to be added. The understanding shows that the path from problem to solution is one which incrementally adds solution knowledge as well as may span into distinct alternatives at each stage.

In design, a problem is usually one of offering functionality. Examples include the functionality of protecting against cold weather and the functionality of defining a space for certain activities [69]. A solution to the problem is the identification of the characteristics of artefacts which offer such functionality.

The notion of functionality is to be understood in a broad sense here. Requirements of buildings — defining the design problem at hand — can be categorized into three distinct sorts: functional, spatial and contextual.

Definition 5.4 (Functional requirements) By functional requirements, we understand requirements to the behaviour of an artefact when given certain situation.

A functional requirement of a wall can be for it to reduce noise with a certain percentage. The problem is to fulfil the functional requirements and the solution is a certain set of properties such that it is the case.

Definition 5.5 (Spatial requirements) By spatial requirements, we understand a measurement of the minimum space needed for the activities for which a building is to be built.

A spatial requirement of an office building can be for it to possess 32 office rooms of (at least) $20m^2$ each. The solution is a certain partition and distribution of available space such that it is the case.

Definition 5.6 (Contextual requirements) By contextual requirements, we understand the bindings of the building to the surroundings, which restrict the choices of locations.

A contextual requirement of a kindergarten building is that it is placed at least $240m$ from any highway. The solution is a selection of a certain available building site such that it is the case.

In a sense, a solution to each sort of requirement makes the artefact satisfy certain needs. Functional requirements are met by ascribing the right sets of properties to objects. Spatial requirements are met in a similar manner, though with a focus on spatial properties like length and height, and in addition by including spatial–oriented relations between objects. Contextual requirements usually concern the relations in which an artefact is supposed to stand to the surroundings. Thus, it is not primarily concerned with the artefact in isolation. Still, certain restrictions due to the surroundings may influence the problem solving processes concerning functionality and spatiality. E.g. there may be restrictions which say that the building should fit by not being too tall, a property which involves surrounding buildings.

Various design and construction technical conceptions comply with the philosophical and scientific understanding of problem and solution. One of the most

prominent within architecture, engineering, and construction (AEC), is included in Gielingh's definition of *The General AEC reference model*; also known as the *GARM* [89]. In the GARM, artefacts are specified in terms of *Product Definition Units* (*PDU*). By a PDU, Gielingh understands any part of an artefact interesting or important enough to record information about. Seven life–cycle stages of artefact development are distinguished; all considered sub–types of PDUs. Two of these are *Functional Unit* and *Technical Solution*.

A functional unit is a product definition unit concerned with what is required of the artefact; also called "*as–required*". A Technical solution is a product definition unit concerned with how the requirement, stated by the functional unit as a problem is solved; also called "*as–designed*".

The GARM model was proposed for inclusion in the STEP standard [6]. Even though it was never accepted, it has had a tremendous influence on research and on industry practice within standardisation of product data, interoperability, etc.

5.2.2 Design move

Adding objects, properties of objects and relations between objects to a design configuration makes the design more specialised and thus more narrow for interpretation. The actions conceptually change a design and are what we shall understand by *design moves*.

The notion of design moves (and its varieties) was originally introduced by Schön [149] and used as fundamental notion in [164, 139, 79].

Definition 5.7 (Design move (cognitively)) By a design move, we understand an action of thought with which the practitioner conceives a design aspect or changes a design conceptually.

We shall consider design moves to be formally defined functions which change a design model — from one stage into another.

In incremental design, the various stages are related by design moves. Recording the moves and structuring design stages requires a proper and formal definition of the relation. More importantly, the definition must explicitly state how incrementality is incorporated such that we can talk of design specialisation. This means, that the relation between design stages is an ordering relation.

The intuition of designing being a series of choices and changes to a design configuration outlines a hierarchy of design stages.

In the following sections, we consider two definitions of the relation between design stages: (i) The approach of *class partition*, and the approach of *Design Lattices*. The two approaches differ in the sense that the former is based on a principle of class partition, in which each class represents possible realisations of the design in question. As a general term we shall use the term "*individual*" to denote such realisations. A collection of such individuals makes the extension of a class[4]. Design lattices, on the other hand, represent dual knowledge to individuals, like the properties of individuals. Also, design lattices facilitate multiple inheritance of objects, properties of objects, and relations between objects. Both approaches are fundamental in the sense that they are founded on mathematical abstractions: *trees* and *lattices*, respectively.

5.3 Class partition

The approach of class partition is based on the notion of classification. In the beginning of the design exploration process, we have a concept under which the possible designs as individuals fall[5]. A collection of individuals is called a class.

A full collection of all possible individuals falling under a concept is called the *extension* of the concept. These may, however, not all qualify as proper solutions to the design problem at hand.

The exploration process now divides the class recursively until a satisfactory design concept is reached. In mathematics this is called *class partition*, or simply *partition*.

Definition 5.8 (Class partition) By a class partition, in mathematics, we understand a division of a set of individuals into mutually exclusive and jointly exhaustive subsets. Each such subset is a class of which individuals are intimately associated by means of an equivalence relation [8].

Instead of expressing equivalence relations, it is common to describe individuals of classes by expressions with a predicative nature on individuals. Such ex-

[4] although not all collections may be proper class extensions.

[5] Several philosophical problems are here present. Primarily, we have the problem of having collections of artefacts not yet existing. Suggestions for solving this problem include applying a Platonic perspective [84] and introducing possible worlds (see the papers in Chapter 8 and Chapter 9).

pressions may refer to properties of the individuals which means that what is expressed is the conditions for individuals to belong to a certain class.

The structure of class partitions spans a tree structure as depicted on Figure 5.2.

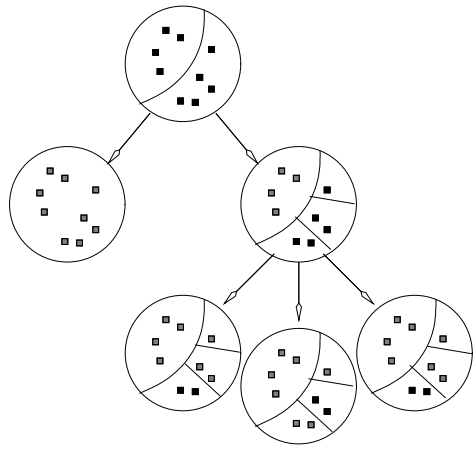


Figure 5.2: Classification of individuals.

Design choices here means selecting a certain path among possible alternatives. Each choice decreases the space of individuals and the process stops when a satisfactory design concept is reached. A partition of the class of bridges is pictorially shown on Figure 5.3[6].

It is fundamental to the approach, that what is partitioned right from the start is the class of all individuals (perhaps restricting to some basic concept like house or hospital).

Model–theoretically, the approach is extensional as each stage in the design process associates to it a class of individuals. Going from problem to solution means going from the class of all individuals to more and more narrow classes by means of restriction. Each such restriction corresponds to the incrementally added conditions derived from the design problem.

Although the partition approach complies with the problem–solution understanding and the mathematical notion of class partition, we claim six objections to it.

[6]Note, that any pictorial representation of concepts which differ in abstraction is wrong. Excluding parts of a picture does not correctly express the fact that what is presented is more abstract. However, this is the only graphical means for expression possible.

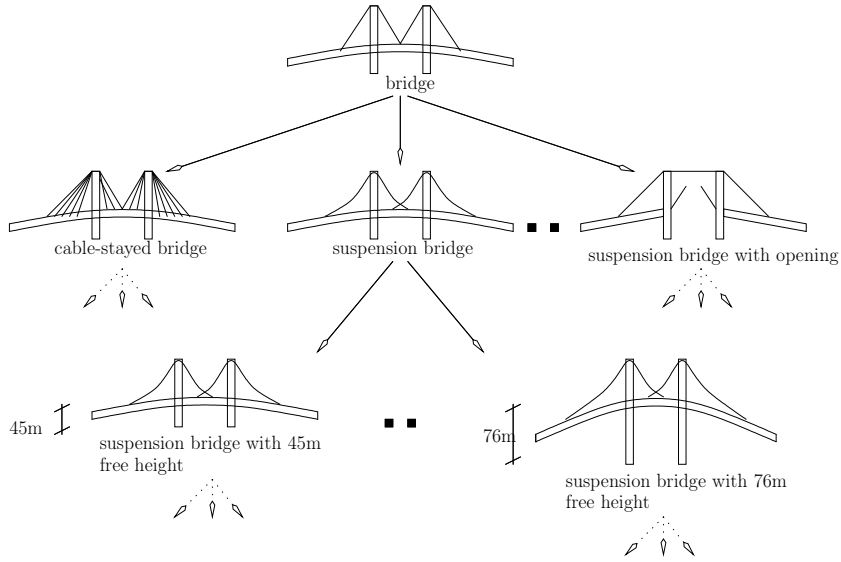


Figure 5.3: Partition of the bridge class.

The first objection is that it may not be possible always to state the overall concept under which a final design solution falls. As argued by Schön [149], working with a design may open for other sorts of solutions than first conceived. Thus, a straight top–down approach is not always possible. As a rule of thumb, we could say that the better we know the domain of discourse, the better can a top–down approach be applied. Solving this problem by starting with a very general concept, may result in loss of the focus instead. One of the characteristic of design is that we do not really know the class of design solutions that well — especially not if we are talking about innovative design; i.e. design of completely new sorts of artefacts.

The second objection is that we are hardly interested in representing the classes of design individuals which have not been selected. Thereby, the whole idea of class partition seems to be lost. In some cases, a design process will then be a series of class name definitions of which the definiendum may be difficult to express simply because we need to express what individuals the classes do not include.

The third objection is that the approach does not give a good account on how to handle removal of design knowledge. We cannot overlook the fact that some design moves are in fact removal of objects, properties of objects or relations between objects, from a design. Basically, there are two problems. The first is that in order to maintain consistency, removal of such design information has

to be performed in the opposed order in which it has been added. However, removing one piece of information may also remove other pieces of information, depending on the order in which the classes have been partitioned. In general, we may need to make several such steps upwards the classification structure before the unintended design information has been removed. Along the way we may have removed other design information which we wished to keep. The problem, we call the *backtracking problem*. The second problem appears if we try to solve the first by allowing removal downwards the tree structure. This may result in obscure situation, in which we start with a general design concept and ends up with this concept again because restrictions have been removed.

The fourth objection is that identical pieces of design information may exists in stages of non–equivalent classes in a partition. This means that we have redundant information. Redundancy, potentially is the cause of inconsistency problems, and should in general be avoided.

The fifth objection is that the partition approach do not offer convenient and easy ways of combining distinct, possibly overlapping design solutions.

The sixth objection is that it may be a problem categorising a design individual to belong to a certain class. In knowledge engineering, it is common to categorise individuals according to their properties. However, the individuals we are considering may consist of several objects; each having own properties. Classifying such multi–object constellations is complicated. The reason is that we need to include conditions for whether design concepts have parts of certain kinds. Thereby, the problem of classification becomes a type identity problem based on part–whole relations of objects. This problem is one of the problems in part–whole theory (in general, *extensional mereology*) for which objections are strong (cf. Chapter 8).

Also, other concept names may better characterise intention and use of the individuals of a class. E.g. several theaters are housed in closed factory buildings.

The first and second objections indicate that the design process should be considered to start with an empty configuration. To this empty configuration, design knowledge is added incrementally, thereby specialising the design and decreasing the solution space. Instead of having concepts for each design stage, we suggest that we focus on properties of objects and relations between objects.

The third, fourth, and fifth objections call for a distinction between two sorts of design moves: (i) those which explore design alternatives, and (ii) those which combine design alternatives. We call these: design move *by aspect* and design move *by configuration*, respectively. The distinction is convenient in order to represent the many–sorted knowledge of designs.

Together, the five objections suggest that a convenient structure for recording and structuring design processes, is a lattice.

The sixth objection raises more philosophical and epistemological questions of how names can be used to denote classes of artefacts. Among these is the question of whether we can relate concept and use/rôle of objects falling under concepts. However, we shall not treat this issue in this paper (cf. Chapter 8).

5.4 Design lattices

The approach of design lattices does not rely on concepts for denoting individual design solutions. Rather it relies on the properties which aim at characterising the objects in design solutions. The approach is thus intensional, whereas the other is extensional.

Design lattices have the structure of mathematical lattices.

Definition 5.9 (Design lattice) By a design lattice, we understand a collection of design configurations which are partially ordered. Each design configuration corresponds to a specific stage in the design process. The relation stands between such design stages corresponds to design moves which specialise the configurations.

Figure 5.4 shows a design lattice for one design process leading to a bridge design similar to the one depicted on Figure 5.3.

A design lattice is always bounded upwards. The bound is called Top ($\top$) and represents the empty design configuration; that which has the empty set of objects. From Top, we can go to stages such that objects, properties of objects, and relations between objects are added — one step at a time, though. Design configurations (i.e. stages) are brought together by means of the lattice operation *meet*. This operation takes two design configurations (partial designs) and gives the configuration with the union set of objects and the union set of properties for equally named objects. A similar account applies to the relations in the two models.

It is not inevitable that design lattices are bound downwards. The reason is that lattice meet can be applied for later being abandoned in favour of the meet of other design stages. However, a design lattice can always be bound downwards by adding to it lattice meet of all lower branches.

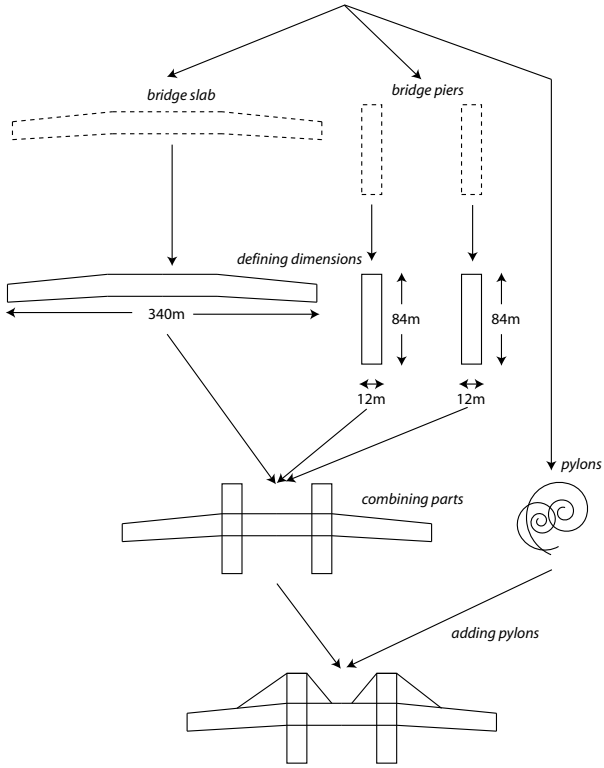


Figure 5.4: Design lattice leading to a bridge design.

As a general interpretation rule of design stages in design lattices, we can say that objects in a design are required to be existing parts of the artefact in mind. Similar goes for properties and relations.

Model–theoretically, design lattices have similarities with the approach of class partition. The empty configuration of a design lattice corresponds to a most general design concept; the concept for which we do not require that individuals consist of certain objects nor that certain properties are ascribed the objects. All artefacts satisfy these criteria. Adding objects, properties of objects, and relations between objects introduce restrictions which specialise the design concepts. Such concepts could be the concept of individuals consisting of two objects of which the former is ascribed the property of being made of steel and the latter the property of being made of wood. Suitable concept names may be difficult to define but with design lattices, we do not have to do so.

Compositionally, the notion of design lattices have similarity with extensional

mereology — i.e. the theory of part–whole relations. The empty configuration corresponds to *absurdum* — the empty object constellation. Lattice meet of all design configuration corresponds to *universe*; although in a very local understanding. In mereology universe is the mereological sum of all objects in the world. In this doctrine, objects are either atomic or sums of other objects, where sum is the inverse operation of performing a partition of the universe or a part of the universe. However, mereology focuses on the opposed direction of going from empty configurations with its emphasis on universe.

In order to overcome the problems stated by the six objections, design lattices make a distinction between two sorts of design moves: design moves *by aspect* and design moves *by configuration*.

From lattice theory, we know that redundancy can be eliminated by means of restructuring. This is a possibility, but not a requirement, in design lattices as we aim at recording the design decisions of the practitioner. The partial ordering of lattices ensures that consistency is maintained. Figure 5.5 shows how design lattices handle object removal compared to an approach which does not maintain order of design stages.

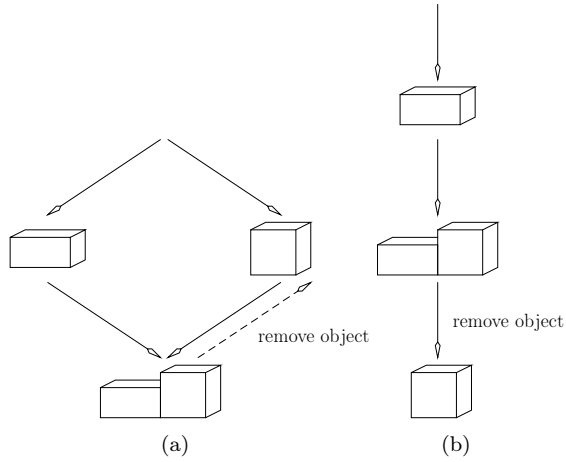


Figure 5.5: Object removal: Design lattice (a), without order (b).

However, it should be stated that design lattices do not give complete freedom in backtracking. Consider the situation where a set of objects have been put together with lattice meet, and then from this stage additional objects are added. Removing one of the first added objects cannot be done from the resulting stage; only from the stage of the first meet operation. That is, the backtracking problem *can* occur with design lattice, but they still add more flexibility (with

respect to order consistency) than the class partition approach.

5.4.1 Design choice by aspect

Design choice *by aspect* is the class of design moves which add an object, a property of an object, or a relation between two objects, to a design. The choice is made between different incomparable sorts of design knowledge; that is: objects, properties of various domains, and relations of various kinds. The domains of such knowledge are called aspects. That is:

Definition 5.10 (Aspect) By an aspect, we understand the conception of an object or a domain of properties/relations which relate by their kinds.

In a design lattice, design choice by aspect means selecting a certain design path. That is, exploring possibilities of a certain domain of properties, like colours or material. Figure 5.6 shows the principle of design choice by aspect.

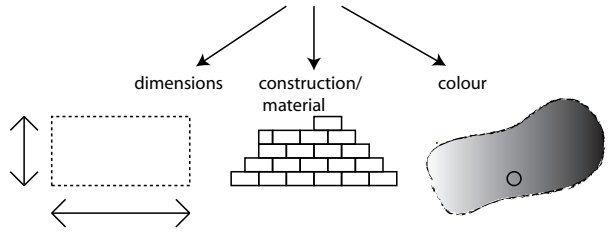


Figure 5.6: Design choice by aspect.

The notion of aspect is inspired by Gärdenfors' notion of domains. Besides making partitions of design knowledge into incomparable sorts, design choice by aspect also serves the purpose of making partitions of single property domains. Not all equivalence classes of a partition need to be represented in the lattice, as indicated on Figure 5.7. Consider a partition of the property of having a length of $3m$ or more into the property of having a length between 3 and $8m$, and 5 and $10m$, respectively. This partition excludes the possibility of having a length of more than $10m$ and less than $3m$. It may seem strange mathematically, but there is no rationality in representing design knowledge which will never be considered to be added to the current design. Selecting the colour green for an object does not commit us to also representing all other colours. We simply represent nodes that correspond to actual or previous design stages.

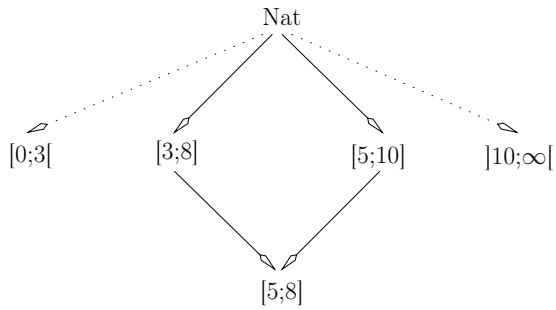


Figure 5.7: Partition of length properties.

5.4.2 Design choice by configuration

Design choice *by configuration* is the class of design moves which combine existing design stages. That is, they combine different aspects into a configuration. For properties, this principle is known as *multiple inheritance* and — concerning representation — it is the point where lattices put distance to trees. For comparable properties like having the colour red and green, the result is the intersection set of the property values. We shall later see how names (called attributes) are used in the identification of property domains.

For incomparable properties like being made of concrete and being shaped as a column, the result is the union set of the property values. An explanation of incomparability is as follows. Consider the property of being transparent and the property of having the colour blue. These properties do not exclude each other; they can co–exist for an object. The reason is that the former is defined as materials capability to let through light, whereas the latter is defined as the property of how light is reflected. The two properties thus instantiate distinct causal relations between the objects in question, and the surrounding. This understanding of properties is adopted from Shoemaker [153] (see also Chapter 9).

Figure 5.8 shows the principle of design choice by configuration.

5.5 Ontological entities for design representation

We search for a minimal set of ontological entities to which references can be made in design descriptions. This set seems to include the notions of objects,

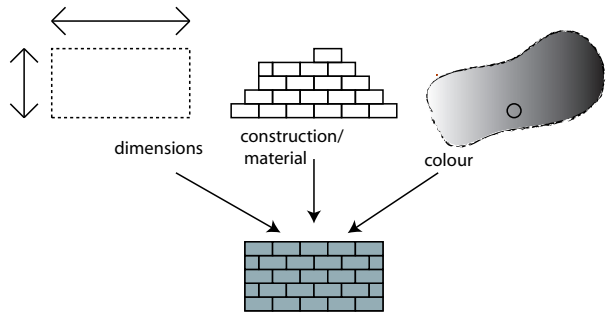


Figure 5.8: Design choice by configuration.

properties, and relations.

5.5.1 Objects as containers

In research of conceptual design systems, a focus has been directed towards the notion of properties [64, 82]. In this paper, we shall maintain this focus. Thereby, we follow the property oriented approach suggested in [62] (Chapter 4) which has many similarities with the class oriented approach presented in [87].

However, we cannot ignore objects in favour of properties. That would soon present the problem of formally distinguishing two parts of an artefact which have been ascribed identical properties but cognitively represents distinct object conceptions. An example is two identical windows. In description of the windows, we might need references to infinite many properties (of which most would be of little importance) in order to distinguish the two objects.

Therefore, we introduce objects as a kind of containers. They are simply identifiers which facilitate distinguishing equal sets of property and relations, respectively. They can be thought of as containers in which we can fill in representations for properties.

5.5.2 Properties

Properties are refereed to when describing or characterizing objects formally and in every days life. First, the notion of properties is included in knowledge engineering in order to distinguish objects and to define similarity of objects. A similar aim can be seen in the metaphysical world in which the notion of property

relates to perception, characterisation, and communication, as well as causes and effects (causation). Some basic work about properties as metaphysical entities, include [71, 143, 122, 153, 86].

Gärdenfors argues in [86] for making a distinction between properties and regions of properties (property domains), and concepts. The latter is a combination of properties from various domains. Gärdenfors' view adds an extra dimension to ontologies coming out of analytic philosophy. Important examples are Frege's introduction of truth–conditions and Russell's notion of universals [71, 143].

In this paper, we shall understand property designations as pairs. The first component of such a pair is a name of the domain to which the property belongs. We shall call this component the *attribute*. The second component of the pair is a set of values.

The question is now how we should understand properties ontologically. Most important: What sorts of property values should we include? Shoemaker [153, 152], following Geach [88], chooses to exclude extrinsic properties, and thus only focus on intrinsic properties.

> Extrinsic properties are the properties that objects possess in virtue of a connection to their surroundings. Intrinsic properties are the properties that objects possess completely independently of any surroundings.

Shoemaker makes his choice in his epistemological search for an explanation of how we can know that an object has certain properties. This becomes interesting when analysing design rationality in context of the relation to functional requirements (see Chapter 9).

Shoemaker thus concentrates on properties that are *genuine*. The broad notion of *Cambridge properties* satisfies the criteria that loosing or gaining such a Cambridge property results in a *Cambridge change*[7] — simply that Fx at time t and $\neg Fx$ at time $t' > t$. However, this definition also applies to so–called *Mere–Cambridge* properties like for a person to be 100 miles from a burning barn. That is, the class of *Mere–Cambridge* properties is a sub–class of the class of extrinsic properties. According to Shoemaker, there is hardly a good metaphysical reason for maintaining extrinsic properties. Being 100 miles from a burning barn is simply a topological relation between two objects; not something which is intrinsically part of the identity of objects[8].

[7] In [8] though defined more restrictively; namely as a *non–genuine change*.

[8] However, relations (especially part–whole relations) may be important in order to designate objects. We argue for this perspective in Chapter 8

We claim that in order for an artefact description to be natural, knowledge of how objects relate must be expressed by reference to relations and not to extrinsic properties. Thereby, we follow Shoemaker's distinction between properties solely associated with an object, and properties relying on the existence of other objects. The latter, we — as Shoemaker — exclude all together and replace with binary relations.

5.5.3 Relations

In our understanding, relations differ from properties in that their instantiation depend on the declaration of two objects. In this work, we shall settle with binary relations. A relation $R(x, x')$ reads: Object x has relation R to x'. The relation is not necessarily symmetric.

Topological relations, which relate objects spatially, are inevitable in all design descriptions of artefacts. E.g., a topological relation can state that two objects are placed within a certain distance, on top of each other, one approaching another with some angle, etc.

As for properties, we understand relation designations as pairs of which the first component is an attribute and the second is a set of values. Relation attributes and values may, however, distinct from those of properties.

Topological relations define orientation for the included objects in a model and we thus do not need any coordinate system to help us. We see coordinate systems as convenient when describing something geometrically. However, conceptually they are a flaw of design system, although interpretation of conceptual models may require some kind of translation from relation designations to geometrical data (see also our discussion on this subject in Section 7.10.1 in Chapter 7).

We believe — as Shoemaker [153, 152] — that information like placement in space is not a property of the object but rather a relation between objects.

5.5.4 Values and value domains

Consider the property of having a length of $5m$ and the property of having a length of more than $3m$. In Formal Concept Analysis, we need to express these as unary predicates as subsumption is not defined between the individual properties they denote. We would have to write length_5m and length_more_than_3m which is a standard solution to the problem of *many–valued logic* [85]. Both

names have predicative nature; however, in design we need to handle properties of object, isolated from any formal context. In Formal Concept Analysis, a formal context is what defines extension and intension of a concept. It is basically a table of which the columns can be represented by predicates and the rows correspond to objects being predicated. The cells within the table are Boolean values. Formal contexts are most often used in bottom up data analysis which is a process different from the constructive process of designing. If we were to compare the properties denoted by the predicates like the above, the comparison would be merely syntactical. We seek a more smooth approach for our purpose. Therefore, we have introduced attributes and values for designation of properties and relations. It is important to emphasize, that the attribute of a property should have predicative nature — not on objects — but on property values. E.g. colour is predicative on the values red and green, and denotes the ontological domain to which such values belong. That an object has a property, which is defined as a set with one property value, means that the object is considered or expected to have that specific property. If the set of property values contains more than one value, each value represents a possible or acceptable value for that property. In case of the empty set, we talk of the absurd property, which may be used to represent invalid configurations or conflicts.

5.6 Artefact models

An artefact model (θ:Θ) is a record of two maps stating: (i) The bindings of objects (x:X) to property sets (ps:PS), and (ii) the bindings of object pairs to binary relation sets (rs:RS). A property set (ps:PS) is a map from property attributes (a:AP) to value sets (vs:VP). A relation set (rs:RS) is a map from relation attributes (a:AR) to value sets (vs:VR). Note, that we make a distinction between values for properties and values for relations.

We say that an artefact model is wellformed if and only if: (i) all related objects are in the domain of the map that binds objects to property sets. Formally, we write:

type
Θ' :: objects:X $\xrightarrow[m]{}$ PS relations:(X × X) $\xrightarrow[m]{}$ RS,
$\Theta = \{|\theta{:}\Theta' \bullet \mathrm{wf}(\theta)|\}$,
$\mathrm{PS}' = \mathrm{AP} \xrightarrow[m]{} \mathrm{VP}$**-infset**,
$\mathrm{PS} = \{|\mathrm{ps}{:}\mathrm{PS}' \bullet (\forall\ \mathrm{vs}{:}\mathrm{VP}$**-infset** $\bullet\ \mathrm{vs} \in$ **rng** ps $\Rightarrow\ \mathrm{vs} \neq \{\})|\}$
$\mathrm{RS}' = \mathrm{AR} \xrightarrow[m]{} \mathrm{VR}$**-infset**,
$\mathrm{RS} = \{|\mathrm{rs}{:}\mathrm{RS}' \bullet (\forall\ \mathrm{vs}{:}\mathrm{VR}$**-infset** $\bullet\ \mathrm{vs} \in$ **rng** rs $\Rightarrow\ \mathrm{vs} \neq \{\})|\}$
AP,

```
  AR,
  VP,
  VR

value
  wf: Θ′ → Bool
  wf(θ) ≡
    (∀ x,x′:X • ((x,x′) ∈ dom relations θ ∨ (x′,x) ∈ dom relations θ) ⇒
      {x,x′} ⊆ dom objects θ),
```

5.6.1 Design moves on artefact models

Whether design is understood as spanning a lattice, a tree, or some other structure, the relation between design representations is crucial.

Definition 5.11 (Design move (formally)) A *design move* is a transition from one artefact model (θ:Θ) to another artefact model (θ':Θ); denoted $\psi(\theta) \equiv \theta'$.

A design move $\psi(\theta) \equiv \theta'$ is valid, if and only if θ' is a model of an artefact idea which is a *relevant successor* to the artefact idea corresponding to the model θ. The notion of relevant successor has been defined by Galle [79, 80]. However, we need a more mathematical explicit definition for our formal purpose here. We recognize that the definition of *relevant successor* comply with extensional semantics: Each model denotes the possibly infinite set of artefacts which complies with the restrictions and existential commitment expressed in the model.

However, trouble is present when we wish to handle distinct objects which have their own set of properties and relations. In our approach a model b is a relevant successor of a model a, if an only if b includes the objects, properties of objects and relations of a. This means that the relation between artefact models is not a ordinary *kind–of* relation as for property subsumption. We write $\theta' \leq \theta$ to emphasize the restriction on value domains downwards the ordering of artefact models. Especially, $\forall\theta \bullet \theta \leq e$, where a and e are artefact models and e is the empty model.

5.7 Partial order

As a first step towards design lattices, we define the relation $(\leq,\Theta)$ between artefact models.

value
$\leq$: $\Theta \times \Theta \rightarrow$ **Bool**

axiom $\forall\ \theta, \theta'{:}\Theta \bullet$
$\theta \leq \theta' \equiv$
(∀ x:X • x ∈ **dom** objects(θ') ⇒ (x ∈ **dom** objects(θ) ∧
(∀ a:AP • a ∈ **dom** objects(θ')(x) ⇒
(a ∈ **dom** objects(θ)(x) ∧
objects(θ)(x)(a) ⊆ objects(θ')(x)(a))))) ∧
(∀ (x,x′):(X×X) • (x,x′) ∈ **dom** relations(θ') ⇒
((x,x′) ∈ **dom** relations(θ) ∧
(∀ a:AR • a ∈ **dom** relations(θ') ⇒
(a ∈ **dom** relations(θ)(x,x′) ∧
relations(θ)(x,x′)(a) ⊆ relations(θ')(x,x′)(a)))))

Design moves are either single value functions ($\psi{:}\Psi$), or the binary lattice operations join ($\sqcap$) or meet ($\sqcup$). Design moves must preserve the relation $(\leq,\Theta)$:

type
$\Psi = \Theta \rightarrow \ldots \rightarrow \Theta$,
$\sqcap$: $\Theta \times \Theta \rightarrow \Theta$,
$\sqcup$: $\Theta \times \Theta \rightarrow \Theta$

axiom $\forall\ \theta, \theta'{:}\Theta, \psi{:}\Psi \bullet$
$\psi(\theta) \leq \theta$,
$\theta \leq \sqcap(\theta,\theta')$,
$\theta' \leq \sqcap(\theta,\theta')$,
$\sqcup(\theta,\theta') \leq \theta$,
$\sqcup(\theta,\theta') \leq \theta'$

With "..." we intend to express that the function may take additional parameters.

Definition 5.12 (Partial ordering) From [85] and [13] we have that a binary relation is a partial ordering if it satisfies the criteria of *reflexivity*, *anti–symmetry*, and *transitivity*:

$$\theta \leq \theta \qquad \textit{Reflexivity} \qquad (5.1)$$

$$\theta \leq \theta' \wedge \theta \neq \theta' \Rightarrow \sim (\theta' \leq \theta) \qquad \textit{Anti–symmetry} \qquad (5.2)$$

$$\theta \leq \theta' \wedge \theta' \leq \theta'' \Rightarrow \theta \leq \theta'' \qquad \textit{Transitivity} \qquad (5.3)$$

In the following, we shall utilize the following lemma:

Lemma 5.13 *The two mappings named* objects *and* relations *in the type of artefact models (θ:Θ), have structures that are isomorphic.*

Proof: *Substitute $[z/x]$ and $[z/(x, x')]$, respectively, throughout the definition of $\leq$. Furthermore, replace types PS with RS, AP with AR, and VP with VR, and vice versa.*

From the Lemma 5.13, we have that proofs for partial ordering and lattice equations only need to be performed for the mapping *objects* in artefact models. Then the proofs apply to the mapping *relations* as well.

Proposition 5.14 *The binary relation ($\leq$,Θ) is* reflexive.

Proof: We show that $\theta \leq \theta$ holds. From Lemma 5.13 and the definition of $\leq$, we have:

$$\begin{array}{l}(\forall\ \text{x:X} \bullet \text{x} \in \textbf{dom}\ \text{objects}(\theta) \Rightarrow (\text{x} \in \textbf{dom}\ \text{objects}(\theta) \wedge \\ \quad (\forall\ \text{a:AP} \bullet \text{a} \in \textbf{dom}\ \text{objects}(\theta)(\text{x}) \Rightarrow \\ \quad\quad (\text{a} \in \textbf{dom}\ \text{objects}(\theta)(\text{x}) \wedge \text{objects}(\theta)(\text{x})(\text{a}) \subseteq \text{objects}(\theta)(\text{x})(\text{a})))))\end{array}$$

As $(z \subseteq z) \equiv \textbf{true}$:

$$\begin{array}{l}\equiv (\forall\ \text{x:X} \bullet \text{x} \in \textbf{dom}\ \text{objects}(\theta) \Rightarrow (\text{x} \in \textbf{dom}\ \text{objects}(\theta) \wedge \\ \quad\quad (\forall\ \text{a:AP} \bullet \text{a} \in \textbf{dom}\ \text{objects}(\theta)(\text{x}) \Rightarrow \\ \quad\quad\quad (\text{a} \in \textbf{dom}\ \text{objects}(\theta)(\text{x}) \wedge \textbf{true}))))\end{array}$$

As $z \wedge \textbf{true} \equiv \textbf{true}$:

$$\begin{array}{l}\equiv (\forall\ \text{x:X} \bullet \text{x} \in \textbf{dom}\ \text{objects}(\theta) \Rightarrow (\text{x} \in \textbf{dom}\ \text{objects}(\theta) \wedge \\ \quad\quad (\forall\ \text{a:AP} \bullet \text{a} \in \textbf{dom}\ \text{objects}(\theta)(\text{x}) \Rightarrow \text{a} \in \textbf{dom}\ \text{objects}(\theta)(\text{x}))))\end{array}$$

As $(z \Rightarrow z) \equiv \mathbf{true}$:

$\equiv (\forall$ x:X • x $\in$ **dom** objects(θ) $\Rightarrow$ (x $\in$ **dom** objects(θ) $\wedge$ **true**))

As $(z \wedge \mathbf{true}) \equiv \mathbf{true}$:

$\equiv (\forall$ x:X • x $\in$ **dom** objects(θ) $\Rightarrow$ x $\in$ **dom** objects(θ))

As $z \Rightarrow z \equiv \mathbf{true}$:

$\equiv$ **true**

QED

□

Proposition 5.15 *The binary relation* $(\leq, \Theta)$ *is* symmetrical.

Proof: We show that $\theta \leq \theta' \wedge \theta \neq \theta' \Rightarrow \sim (\theta' \leq \theta)$. *We assume* $\theta \Rightarrow \theta' \wedge \theta \leq \theta' \wedge \theta \neq \theta'$ *and prove that* $\theta' \leq \theta$ *yields a contradiction.*

From Lemma 5.13 and the definition of $\leq$, *we have:*

$(\forall$ x:X • x $\in$ **dom** objects(θ) $\Rightarrow$ (x $\in$ **dom** objects(θ') $\wedge$
  ($\forall$ a:AP • a $\in$ objects(θ)(x) $\Rightarrow$
    (a $\in$ objects(θ')(x) $\wedge$ objects(θ')(x)(a) $\subseteq$ objects(θ)(x)(a)))))

From the assumption we derive (in the quantification of x and a) that:

objects(θ)(x)(a) $\subseteq$ objects(θ')(x)(a) $\wedge$ objects(θ)(x)(a) $\neq$ objects(θ')(x)(a) $\equiv$
  objects(θ)(x)(a) $\subset$ objects(θ')(x)(a)

Thereby, we have:

$(\forall$ x:X • x $\in$ **dom** objects(θ) $\Rightarrow$ (x $\in$ **dom** objects(θ') $\wedge$
$(\forall$ a:AP • a $\in$ **dom** objects(θ)(x) $\Rightarrow$
(a $\in$ **dom** objects(θ')(x) $\wedge$ **false**))))

As $z\wedge$ **false** $\equiv$ **false**:

$\equiv (\forall$ x:X • x $\in$ **dom** objects(θ) $\Rightarrow$ (x $\in$ **dom** objects(θ') $\wedge$
$(\forall$ a:AP • a $\in$ **dom** objects(θ)(x) $\Rightarrow$ **false**)))

From the assumption, we derive (in the quantification of x and a) that:

objects(θ)(x)(a) $\subseteq$ objects(θ')(x)(a) and objects(θ)(x)(a) $\neq$ objects(θ')(x)(a)

$\equiv$ objects(θ)(x)(a) $\subset$ objects(θ')(x)(a)

This means that **dom** objects(θ') $\neq \{\}$.

That is: $\exists$ a:AP • a $\in$ **dom** objects(θ')(x).

We have:

$(\forall$ x:X • x $\in$ **dom** objects(θ) $\Rightarrow$ (x $\in$ **dom** objects(θ') $\wedge$ **false**))

As ($z\wedge$ **false**) $\equiv$ **false**:

$\equiv (\forall$ x:X x $\in$ **dom** pm $\Rightarrow$ **false**)

From the assumption, we have that **dom** objects(θ) $\neq \{\}$.

That is, $\exists$ x:X • x $\in$ **dom** objects(θ). We have:

$\equiv$ **false**

QED

□

Proposition 5.16 *The binary relation* $(\leq,\Theta)$ *is* transitive.

Proof: We show that $\theta \leq \theta' \wedge \theta' \leq \theta'' \Rightarrow \theta \leq \theta''$. *We assume that* $\theta \leq \theta' \wedge \theta' \leq \theta''$ *and prove that* $\theta \leq \theta''$. *From Lemma 5.13 and the definition of* $\leq$, *we have:*

$$
\begin{array}{l}
(\forall\ \mathrm{x{:}X} \bullet \mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta'') \Rightarrow (\mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta) \wedge \\
\quad (\forall\ \mathrm{a{:}AP} \bullet \mathrm{a} \in \mathbf{dom}\ \mathrm{objects}(\theta'')(\mathrm{x}) \Rightarrow \\
\qquad (\mathrm{a} \in \mathbf{dom}\ \mathrm{objects}(\theta)(\mathrm{x}) \wedge \mathrm{objects}(\theta)(\mathrm{x})(\mathrm{a}) \subseteq \mathrm{objects}(\theta'')(\mathrm{x})(\mathrm{a})))))
\end{array}
$$

From the assumption, we derive (in the quantification of x and a) that:

$$
\begin{array}{l}
\mathrm{objects}(\theta)(\mathrm{x})(\mathrm{a}) \subseteq \mathrm{objects}(\theta')(\mathrm{x})(\mathrm{a}) \wedge \\
\mathrm{objects}(\theta')(\mathrm{x})(\mathrm{a}) \subseteq \mathrm{objects}(\theta'')(\mathrm{x})(\mathrm{a})
\end{array}
$$

$$
\equiv \mathrm{objects}(\theta)(\mathrm{x})(\mathrm{a}) \subseteq \mathrm{objects}(\theta'')(\mathrm{x})(\mathrm{a})
$$

Because of transitivity of set-inclusion, we have:

$$
\begin{array}{l}
\equiv (\forall\ \mathrm{x{:}X} \bullet \mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta'') \Rightarrow (\mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta) \wedge \\
\qquad (\forall\ \mathrm{a{:}AP} \bullet \mathrm{a} \in \mathbf{dom}\ \mathrm{objects}(\theta'')(\mathrm{x}) \Rightarrow \\
\qquad\quad (\mathrm{a} \in \mathbf{dom}\ \mathrm{objects}(\theta)(\mathrm{x}) \wedge \mathbf{true}))))
\end{array}
$$

As $(z \wedge \mathbf{true}) \equiv \mathbf{true}$:

$$
\begin{array}{l}
\equiv (\forall\ \mathrm{x{:}X} \bullet \mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta'') \Rightarrow (\mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta) \wedge \\
\qquad (\forall\ \mathrm{a{:}AP} \bullet \mathrm{a} \in \mathbf{dom}\ \mathrm{objects}(\theta'')(\mathrm{x}) \Rightarrow \mathrm{a} \in \mathbf{dom}\ \mathrm{objects}(\theta)(\mathrm{x}))))
\end{array}
$$

From the assumption, we derive (in the quantification of x and a) that:

$$
\begin{array}{l}
\mathbf{dom}\ \mathrm{objects}(\theta')(\mathrm{x}) \subseteq \mathbf{dom}\ \mathrm{objects}(\theta)(\mathrm{x}) \wedge \\
\mathbf{dom}\ \mathrm{objects}(\theta'')(\mathrm{x}) \subseteq \mathbf{dom}\ \mathrm{objects}(\theta')(\mathrm{x})
\end{array}
$$

$$
\equiv \mathbf{dom}\ \mathrm{objects}(\theta'')(\mathrm{x}) \subseteq \mathbf{dom}\ \mathrm{objects}(\theta'')(\mathrm{x})
$$

Because of transitivity of set-inclusion, we have:

$(\forall\ \mathrm{x{:}X} \bullet \mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta'') \Rightarrow (\mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta) \wedge \mathbf{true}))$

(As $z\wedge$) $\mathbf{true} \equiv z$:

$\equiv (\forall\ \mathrm{x{:}X} \bullet \mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta'') \Rightarrow \mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta))$

From the assumption, we derive (in the quantification of x and a) that:

$\mathbf{dom}\ \mathrm{objects}(\theta') \subseteq \mathbf{dom}\ \mathrm{objects}(\theta) \wedge$
$\mathbf{dom}\ \mathrm{objects}(\theta'') \subseteq \mathbf{dom}\ \mathrm{objects}(\theta')$

$\equiv \mathbf{dom}\ \mathrm{objects}(\theta'') \subseteq \mathbf{dom}\ \mathrm{objects}(\theta)$

Because of transitivity of set-inclusion, we have:

$\equiv \mathbf{true}$

QED

□

THEOREM 5.17 *The relation $(\leq,\Theta)$ defines a partial order.*

Proof*: The proof follows directly from Proposition 5.14, Proposition 5.15, and Proposition 5.16.*

□

5.8 Lattices operations

The next step towards design lattices is to define the two lattice operations *meet* ($\sqcup$) and *join* ($\sqcap$). This is done hierarchically on the mapping structures of artefact models. That is, we overload the operations for property sets (ps:PS) as well as for relation sets (rs:RS).

Definition 5.18 (Lattice meet) The *meet* of two artefact models a and b is the artefact model which is the union set of objects, the union set of properties for each common object, and the union set of relations between common objects. For each property and relation the value sets of common attributes are the intersection sets, respectively.

Definition 5.19 (Lattice join) The *join* of two artefact models a and b is the artefact model which is the intersection set of objects, the intersection set of properties for each common object, and the intersection set of relations between common objects. For each property and relation the value sets of common attributes are the union sets, respectively.

Formally, we write this as:

```
value
  ⊓: Θ × Θ → Θ
  ⊔: Θ × Θ → Θ

  ⊓: PS × PS → PS
  ⊔: PS × PS → PS

  ⊓: RS × RS → RS
  ⊔: RS × RS → RS

axiom ∀ ps, ps′:PS •
  ps ⊓ ps′ ≡
    [ a↦vs | a:AP, vs:VP-infset •
      a ∈ dom ps ∧ a ∈ dom ps′ ∧ vs=ps(a) ∪ ps′(a) ],

  ps ⊔ ps′ ≡
    [ a↦vs | a:AP, vs:VP-infset •
      (a ∈ dom ps ∧ a ∈ dom ps′ ∧ vs=ps(a) ∩ ps′(a)) ∨
      (a ∈ dom ps ∧ a ∉ dom ps′ ∧ vs=ps(a)) ∨
      (a ∉ dom ps ∧ a ∈ dom ps′ ∧ vs=ps′(a)) ],

axiom ∀ rs, rs′:RS •
  rs ⊓ rs′ ≡
      [ a↦vs | a:AR, vs:VR-infset •
        a ∈ dom rs ∧ a ∈ dom rs′ ∧ vs=rs(a) ∪ rs′(a) ],

  rs ⊔ rs′ ≡
        [ a↦vs | a:AR, vs:VR-infset •
```

```
                (a ∈ dom rs ∧ a ∈ dom rs′ ∧ vs=rs(a) ∩ rs′(a)) ∨
                (a ∈ dom rs ∧ a ∉ dom rs′ ∧ vs=rs(a)) ∨
                (a ∉ dom rs ∧ a ∈ dom rs′ ∧ vs=rs′(a))],

axiom ∀ θ, θ′:Θ •
  θ ⊓ θ′ ≡
    mk_Θ([x↦ps | x:X, ps:PS •
        x ∈ dom objects(θ) ∧ x ∈ dom objects(θ′) ∧
          ps=objects(θ)(x) ⊓ objects(θ′)(x)],
       [(x,x′)↦ rs | x,x′:X, rs:RS •
        (x,x′) ∈ dom relations(θ) ∧ (x,x′) ∈ dom relations(θ′) ∧
          rs=relations(θ)(x,x′) ⊓ relations(θ′)]),

  θ ⊔ θ′ ≡
    mk_Θ([x↦ps | x:X, ps:PS •
        (x ∈ dom objects(θ) ∧ x ∈ dom objects(θ′) ∧
          ps=objects(θ)(x) ⊔ objects(θ′)(x)) ∨
        (x ∈ dom objects(θ) ∧ x ∉ dom objects(θ′) ∧ ps=objects(θ)(x)) ∨
        (x ∉ dom objects(θ) ∧ x ∈ dom objects(θ′) ∧ ps=objects(θ′)(x))],
       [(x,x′)↦ rs | x,x′:X, rs:RS •
        ((x,x′) ∈ dom relations(θ) ∧ (x,x′) ∈ dom relations(θ′) ∧
          rs=relations(θ)(x,x′) ⊔ relations(θ′))
       ∨
        ((x,x′) ∈ dom relations(θ) ∧ (x,x′) ∉ dom relations(θ′) ∧
          rs=relations(θ)) ∨
        ((x,x′) ∉ dom relations(θ) ∧ (x,x′) ∈ dom relations(θ′) ∧
          rs=relations(θ′))]),
```

From [85] and [13] we know that a partial ordering is a lattice if it has unambiguous least upper bound and greatest lower bound for all pairs of nodes. We thus need to show the following equations:

$$\theta \leq \theta' \quad \equiv \quad \theta \sqcup \theta' = \theta \tag{5.4}$$

$$\theta \leq \theta' \quad \equiv \quad \theta \sqcap \theta' = \theta' \tag{5.5}$$

Theorem 5.20 *The partial ordering $(\Theta, \leq)$ is a lattice.*

Proof: *We assume $\theta \leq \theta'$ and show $\theta \sqcup \theta' = \theta$. From Lemma 5.13 and the definition of $\leq$, we have:*

```
[x ↦ ps | x:X, ps:PS •
```

$$
\begin{array}{l}
(\mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta) \wedge \mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta) \wedge \\
\quad \mathrm{ps}{=}\mathrm{objects}(\theta)(\mathrm{x}) \sqcup \mathrm{objects}(\theta')(\mathrm{x})) \vee \\
(\mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta) \wedge \mathrm{x} \notin \mathbf{dom}\ \mathrm{objects}(\theta') \wedge \\
\quad \mathrm{ps}{=}\mathrm{objects}(\theta)(\mathrm{x})) \vee \\
(\mathrm{x} \notin \mathbf{dom}\ \mathrm{objects}(\theta) \wedge \mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta') \wedge \\
\quad \mathrm{ps}{=}\mathrm{objects}(\theta')(\mathrm{x}))\,] = \mathrm{objects}(\theta)
\end{array}
$$

From the definition of $\sqcup$ ranging over PS $\times$ PS, we have:

$$
\begin{array}{l}
\equiv [\,\mathrm{x} \mapsto \mathrm{ps} \mid \mathrm{x}{:}\mathrm{X},\ \mathrm{ps}{:}\mathrm{PS} \bullet \\
\quad (\mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta) \wedge \mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta) \wedge \\
\quad\quad \mathrm{ps}{=}[\,\mathrm{a} \mapsto \mathrm{vs} \mid \mathrm{a}{:}\mathrm{AP},\ \mathrm{vs}{:}\mathrm{VP}\textbf{-infset} \bullet \\
\quad\quad\quad (\mathrm{a} \in \mathbf{dom}\ \mathrm{objects}(\theta)(\mathrm{x}) \wedge \mathrm{a} \in \mathbf{dom}\ \mathrm{objects}(\theta')(\mathrm{x}) \wedge \\
\quad\quad\quad\quad \mathrm{vs}{=}\mathrm{objects}(\theta)(\mathrm{x})(\mathrm{a}) \cap \mathrm{objects}(\theta')(\mathrm{x})(\mathrm{a})) \vee \\
\quad\quad\quad (\mathrm{a} \in \mathbf{dom}\ \mathrm{objects}(\theta)(\mathrm{x}) \wedge \mathrm{a} \notin \mathbf{dom}\ \mathrm{objects}(\theta')(\mathrm{x}) \wedge \\
\quad\quad\quad\quad \mathrm{vs}{=}\mathrm{objects}(\theta)(\mathrm{x})(\mathrm{a})) \vee \\
\quad\quad\quad (\mathrm{a} \notin \mathbf{dom}\ \mathrm{objects}(\theta)(\mathrm{x}) \wedge \mathrm{a} \in \mathbf{dom}\ \mathrm{objects}(\theta')(\mathrm{x}) \wedge \\
\quad\quad\quad\quad \mathrm{vs}{=}\mathrm{objects}(\theta')(\mathrm{x})(\mathrm{a}))\,]) \vee \\
\quad (\mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta) \wedge \mathrm{x} \notin \mathbf{dom}\ \mathrm{objects}(\theta') \wedge \\
\quad\quad \mathrm{ps}{=}\mathrm{objects}(\theta)(\mathrm{x})(\mathrm{a})) \vee \\
\quad (\mathrm{x} \notin \mathbf{dom}\ \mathrm{objects}(\theta) \wedge \mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta') \wedge \\
\quad\quad \mathrm{ps}{=}\mathrm{objects}(\theta')(\mathrm{x})(\mathrm{a}))\,] = \mathrm{objects}(\theta)
\end{array}
$$

From the assumption, we have:

$$
\begin{array}{l}
\equiv [\,\mathrm{x} \mapsto \mathrm{ps} \mid \mathrm{x}{:}\mathrm{X},\ \mathrm{ps}{:}\mathrm{PS} \bullet \\
\quad (\mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta) \wedge \mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta) \wedge \\
\quad\quad \mathrm{ps}{=}[\,\mathrm{a} \mapsto \mathrm{vs} \mid \mathrm{a}{:}\mathrm{AP},\ \mathrm{vs}{:}\mathrm{VP}\textbf{-infset} \bullet \\
\quad\quad\quad (\mathrm{a} \in \mathbf{dom}\ \mathrm{objects}(\theta)(\mathrm{x}) \wedge \mathrm{a} \in \mathbf{dom}\ \mathrm{objects}(\theta')(\mathrm{x}) \wedge \\
\quad\quad\quad\quad \mathrm{vs}{=}\mathrm{objects}(\theta)(\mathrm{x})(\mathrm{a}) \cap \mathrm{objects}(\theta')(\mathrm{x})(\mathrm{a})) \vee \\
\quad\quad\quad (\mathrm{a} \in \mathbf{dom}\ \mathrm{objects}(\theta)(\mathrm{x}) \wedge \mathrm{a} \notin \mathbf{dom}\ \mathrm{objects}(\theta')(\mathrm{x}) \wedge \\
\quad\quad\quad\quad \mathrm{vs}{=}\mathrm{objects}(\theta)(\mathrm{x})(\mathrm{a})) \vee \\
\quad\quad\quad \mathbf{false}\,]) \vee \\
\quad (\mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta) \wedge \mathrm{x} \notin \mathbf{dom}\ \mathrm{objects}(\theta') \wedge \\
\quad\quad \mathrm{ps}{=}\mathrm{objects}(\theta)(\mathrm{x})(\mathrm{a})) \vee \\
\quad \mathbf{false}\,] = \mathrm{objects}(\theta)
\end{array}
$$

As $(z) \vee \mathbf{false}\ isz$:

$$
\begin{array}{l}
\equiv [\, \mathrm{x} \mapsto \mathrm{ps} \mid \mathrm{x{:}X},\ \mathrm{ps{:}PS} \bullet \\
\quad (\mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta) \wedge \mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta) \wedge \\
\quad\quad \mathrm{ps}{=}[\, \mathrm{a} \mapsto \mathrm{vs} \mid \mathrm{a{:}AP},\ \mathrm{vs{:}VP}\textbf{-infset} \bullet \\
\quad\quad\quad (\mathrm{a} \in \mathbf{dom}\ \mathrm{objects}(\theta)(\mathrm{x}) \wedge \mathrm{a} \in \mathbf{dom}\ \mathrm{objects}(\theta')(\mathrm{x}) \wedge \\
\quad\quad\quad\quad \mathrm{vs}{=}\mathrm{objects}(\theta)(\mathrm{x})(\mathrm{a}) \cap \mathrm{objects}(\theta')(\mathrm{x})(\mathrm{a})) \vee \\
\quad\quad\quad (\mathrm{a} \in \mathbf{dom}\ \mathrm{objects}(\theta)(\mathrm{x}) \wedge \mathrm{a} \notin \mathbf{dom}\ \mathrm{objects}(\theta')(\mathrm{x}) \wedge \\
\quad\quad\quad\quad \mathrm{vs}{=}\mathrm{objects}(\theta)(\mathrm{x})(\mathrm{a}))\,]) \vee \\
\quad (\mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta) \wedge \mathrm{x} \notin \mathbf{dom}\ \mathrm{objects}(\theta') \wedge \\
\quad\quad \mathrm{ps}{=}\mathrm{objects}(\theta)(\mathrm{x})(\mathrm{a}))\,] = \mathrm{objects}(\theta)
\end{array}
$$

From the assumption, we have that

$$\mathrm{objects}(\theta)(\mathrm{x})(\mathrm{a}) = \mathrm{objects}(\theta)(\mathrm{x})(\mathrm{a}) \cap \mathrm{objects}(\theta')(\mathrm{x})$$

is $\mathrm{objects}(\theta)(\mathrm{x})(\mathrm{a}) \subseteq \mathrm{objects}(\theta')(\mathrm{x})(\mathrm{a})$.

We have:

$$
\begin{array}{l}
\equiv [\, \mathrm{x} \mapsto \mathrm{ps} \mid \mathrm{x{:}X},\ \mathrm{ps{:}PS} \bullet \\
\quad (\mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta) \wedge \mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta) \wedge \\
\quad\quad \mathrm{ps}{=}[\, \mathrm{a} \mapsto \mathrm{vs} \mid \mathrm{a{:}AP},\ \mathrm{vs{:}VP}\textbf{-infset} \bullet \\
\quad\quad\quad (\mathrm{a} \in \mathbf{dom}\ \mathrm{objects}(\theta)(\mathrm{x}) \wedge \mathrm{a} \in \mathbf{dom}\ \mathrm{objects}(\theta')(\mathrm{x}) \wedge \\
\quad\quad\quad\quad \mathrm{vs}{=}\mathrm{objects}(\theta)(\mathrm{x})(\mathrm{a})) \vee \\
\quad\quad\quad (\mathrm{a} \in \mathbf{dom}\ \mathrm{objects}(\theta)(\mathrm{x}) \wedge \mathrm{a} \notin \mathbf{dom}\ \mathrm{objects}(\theta')(\mathrm{x}) \wedge \\
\quad\quad\quad\quad \mathrm{vs}{=}\mathrm{objects}(\theta)(\mathrm{x})(\mathrm{a}))\,]) \vee \\
\quad (\mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta) \wedge \mathrm{x} \notin \mathbf{dom}\ \mathrm{objects}(\theta') \wedge \\
\quad\quad \mathrm{ps}{=}\mathrm{objects}(\theta)(\mathrm{x}))\,] = \mathrm{objects}(\theta)
\end{array}
$$

As $(z \wedge z') \vee (z \wedge \sim z') \equiv z$:

$$
\begin{array}{l}
\equiv [\, \mathrm{x} \mapsto \mathrm{ps} \mid \mathrm{x{:}X},\ \mathrm{ps{:}PS} \bullet \\
\quad \mathrm{x} \in \mathbf{dom} \wedge \mathrm{ps}{=}[\, \mathrm{a} \mapsto \mathrm{vs} \mid \mathrm{a{:}AP},\ \mathrm{vs{:}VP}\textbf{-infset} \bullet \\
\quad \mathrm{a} \in \mathbf{dom}\ \mathrm{objects}(\theta)(\mathrm{x}) \wedge \mathrm{vs}{=}\mathrm{objects}(\theta)(\mathrm{x})(\mathrm{a})\,]\,] = \mathrm{objects}(\theta)
\end{array}
$$

QED

We now assume $\theta \sqcup \theta' = \theta$ and show $\theta \leq \theta'$. From Lemma 5.13 and the definition of $\leq$, we have:

$(\forall\ x{:}X \bullet x \in \mathbf{dom}\ objects(\theta') \Rightarrow (x \in \mathbf{dom}\ objects(\theta) \wedge$
$(\forall\ a{:}AP \bullet a \in \mathbf{dom}\ objects(\theta')(x) \Rightarrow$
$(a \in \mathbf{dom}\ objects(\theta)(x) \wedge objects(\theta)(x)(a) \subseteq objects(\theta')(x)(a)))))$

From the assumption, we derive

$objects(\theta)(x)(a) \cap objects(\theta')(x)(a) = objects(\theta(x)(a)).$

We have:

$(\forall\ x{:}X \bullet x \in \mathbf{dom}\ objects(\theta') \Rightarrow (x \in \mathbf{dom}\ objects(\theta) \wedge$
$(\forall\ a{:}AP \bullet a \in \mathbf{dom}\ objects(\theta')(x) \Rightarrow$
$(a \in \mathbf{dom}\ objects(\theta)(x) \wedge \mathbf{true}))))$

As $(z \wedge \boldsymbol{true}) \equiv z$*:*

$\equiv (\forall\ x{:}X \bullet x \in \mathbf{dom}\ objects(\theta') \Rightarrow (x \in \mathbf{dom}\ objects(\theta) \wedge$
$(\forall\ a{:}AP \bullet a \in \mathbf{dom}\ objects(\theta')(x) \Rightarrow a \in \mathbf{dom}\ objects(\theta)(x))))$

From the assumption, we derive that

$\boldsymbol{dom}\ objects(\theta)(x) \cup \boldsymbol{dom}\ objects(\theta')(x) =$

$\boldsymbol{dom}\ objects(\theta)(x).$

We have:

$(\forall\ x{:}X \bullet x \in \mathbf{dom}\ objects(\theta') \Rightarrow (x \in \mathbf{dom}\ objects(\theta) \wedge \mathbf{true}))$

As $(z \wedge \boldsymbol{true}) \equiv z$*:*

$\equiv (\forall\ x{:}X \bullet x \in \mathbf{dom}\ objects(\theta') \Rightarrow x \in \mathbf{dom}\ objects(\theta))$

From the assumption, we derive that

$\boldsymbol{dom}\ objects(\theta) \cup \boldsymbol{dom}\ objects(\theta') =$

dom *objects(θ).*

We have:

$\equiv$ **true**

QED

We assume $\theta \leq \theta'$ *and show* $\theta \sqcap \theta' = \theta$.

From Lemma 5.13 and the definition of $\leq$, *we have:*

$$
\begin{array}{l}
[\,\mathrm{x} \mapsto \mathrm{ps} \mid \mathrm{x{:}X},\ \mathrm{ps{:}PS} \bullet \\
\quad (\mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta) \wedge \mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta') \wedge \\
\qquad \mathrm{ps}{=}\mathrm{objects}(\theta)(\mathrm{x}) \sqcap \mathrm{objects}(\theta')(\mathrm{x}))\,] = \mathrm{objects}(\theta')
\end{array}
$$

From the definition of $\sqcap$ ranging over PS $\times$ PS, we have

$$
\begin{array}{l}
[\,\mathrm{x} \mapsto \mathrm{ps} \mid \mathrm{x{:}X},\ \mathrm{ps{:}PS} \bullet \\
\quad (\mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta) \wedge \mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta') \wedge \\
\qquad \mathrm{ps}{=}[\,\mathrm{a} \mapsto \mathrm{vs} \mid \mathrm{a{:}AP},\ \mathrm{vs{:}VP}\textbf{-infset} \bullet \\
\qquad\quad \mathrm{a} \in \mathbf{dom}\ \mathrm{objects}(\theta)(\mathrm{x}) \wedge \mathrm{a} \in \mathbf{dom}\ \mathrm{objects}(\theta')(\mathrm{x}) \wedge \\
\qquad\qquad \mathrm{vs}{=}\mathrm{objects}(\theta)(\mathrm{x})(\mathrm{a}) \cup \mathrm{objects}(\theta')(\mathrm{x})(\mathrm{a})\,])\,] = \mathrm{objects}(\theta')
\end{array}
$$

From the assumption, we have that

$$\mathrm{objects}(\theta)(\mathrm{x})(\mathrm{a}) \subseteq \mathrm{objects}(\theta')(\mathrm{x})(\mathrm{a})$$

so

$$\mathrm{objects}(\theta)(\mathrm{x})(\mathrm{a}) \cup \mathrm{objects}(\theta')(\mathrm{x})(\mathrm{a}) = \mathrm{objects}(\theta')(\mathrm{x})(\mathrm{a}).$$

We have:

$$
\begin{array}{l}
[\,\mathrm{x} \mapsto \mathrm{ps} \mid \mathrm{x{:}X},\ \mathrm{ps{:}PS} \bullet \\
\quad (\mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta) \wedge \mathrm{x} \in \mathbf{dom}\ \mathrm{objects}(\theta') \wedge \\
\qquad \mathrm{ps}{=}[\,\mathrm{a} \mapsto \mathrm{vs} \mid \mathrm{a{:}AP},\ \mathrm{vs{:}VP}\textbf{-infset} \bullet \\
\qquad\quad \mathrm{a} \in \mathbf{dom}\ \mathrm{objects}(\theta)(\mathrm{x}) \wedge \mathrm{a} \in \mathbf{dom}\ \mathrm{objects}(\theta')(\mathrm{x}) \wedge \\
\qquad\qquad \mathrm{vs}{=}\mathrm{objects}(\theta')(\mathrm{x})(\mathrm{a})\,])\,] = \mathrm{objects}(\theta')
\end{array}
$$

From the assumption, we have that

dom objects(θ')(x) $\subseteq$ **dom** objects(θ)(x).

We have:

$$[\,x \mapsto ps \mid x{:}X,\ ps{:}PS \bullet \\ (x \in \mathbf{dom}\ \text{objects}(\theta) \wedge x \in \mathbf{dom}\ \text{objects}(\theta') \wedge \\ ps{=}[\,a{\mapsto}vs \mid a{:}AP,\ vs{:}VP\textbf{-infset} \bullet a \in \mathbf{dom}\ \text{objects}(\theta')(x) \wedge \\ vs{=}\text{objects}(\theta')(x)(a)\,]\,]\,] = \text{objects}(\theta')$$

From the assumption, we have that

dom objects(θ') $\subseteq$ **dom** objects(θ)(x).

We have:

$$[\,x \mapsto ps \mid x{:}X,\ ps{:}PS \bullet \\ x \in \mathbf{dom}\ \text{objects}(\theta') \wedge \\ ps{=}[\,a{\mapsto}vs \mid a{:}AP,\ vs{:}VP\textbf{-infset} \bullet a \in \mathbf{dom}\ \text{objects}(\theta')(x) \wedge \\ vs{=}\text{objects}(\theta')(x)(a)\,]\,] = \text{objects}(\theta')$$

QED

We now assume $\theta \sqcap \theta' = \theta$ and show $\theta \leq \theta'$. From Lemma 5.13 and the definition of $\leq$, we have:

$$(\forall\ x{:}X \bullet x \in \mathbf{dom}\ \mathit{objects}(\theta') \Rightarrow (x \in \mathbf{dom}\ \mathit{objects}(\theta) \wedge \\ (\forall\ a{:}AP \bullet a \in \mathbf{dom}\ \mathit{objects}(\theta')(x) \Rightarrow \\ (a \in \mathbf{dom}\ \mathit{objects}(\theta)(x) \wedge \mathit{objects}(\theta)(x)(a) \subseteq \mathit{objects}(\theta')(x)(a)))))$$

From the assumption, we derive that

$$\mathit{objects}(\theta)(x)(a) \cup \mathit{objects}(\theta')(x)(a) =$$

$$\mathit{objects}(\theta')(x)(a).$$

We have:

$\equiv$ *(*$\forall$ *x:X* $\bullet$ $x \in$ **dom** *objects(*θ'*)* $\Rightarrow$ *(*$x \in$ **dom** *objects(*θ*)* $\wedge$
($\forall$ *a:AP* $\bullet$ $a \in$ **dom** *objects(*θ'*)(x)* $\Rightarrow$
($a \in$ **dom** *objects(*θ*)(x)* $\wedge$ **true***))))*

As (z$\wedge$ ***true****)* $\equiv$ *z:*

$\equiv$ *(*$\forall$ *x:X* $\bullet$ $x \in$ **dom** *objects(*θ'*)* $\Rightarrow$ *(*$x \in$ **dom** *objects(*θ*)* $\wedge$
($\forall$ *a:AP* $\bullet$ $a \in$ **dom** *objects(*θ'*)(x)* $\Rightarrow$ $a \in$ **dom** *objects(*θ*)(x))))*

From the assumption, we derive that

dom *objects(*θ*)(x)* $\cap$ ***dom*** *objects(*θ*)(x)* $=$

dom *objects(*θ'*)(x).*

We have:

($\forall$ *x:X* $\bullet$ $x \in$ **dom** *objects(*θ'*)* $\Rightarrow$ *(*$x \in$ **dom** *objects(*θ*)* $\wedge$ **true***))*

As (z $\wedge$ ***true****)* $\equiv$ *z:*

($\forall$ *x:X* $\bullet$ $x \in$ **dom** *objects(*θ'*)* $\Rightarrow$ $x \in$ **dom** *objects(*θ*))*

From the assumption, we derive that

dom *objects(*θ*)* $\cap$ ***dom*** *objects(*θ'*)* $=$

dom *objects(*θ'*).*

We have:

$\equiv$ **true**

QED

$\square$

Artefact models and the associated lattice operations thus satisfy the following lattice axioms:

$$\theta \sqcap \theta = \theta \qquad \textit{idempotency} \tag{5.6}$$
$$\theta \sqcup \theta = \theta \tag{5.7}$$
$$\theta \sqcap \theta' = \theta' \sqcap \theta \qquad \textit{commutativity} \tag{5.8}$$
$$\theta \sqcup \theta' = \theta' \sqcup \theta \tag{5.9}$$
$$\theta \sqcap (\theta' \sqcap \theta'') = (\theta \sqcap \theta') \sqcap \theta'' \qquad \textit{associativity} \tag{5.10}$$
$$\theta \sqcup (\theta' \sqcup \theta'') = (\theta \sqcup \theta') \sqcup \theta'' \tag{5.11}$$
$$\theta \sqcap (\theta \sqcup \theta') = \theta \qquad \textit{absorption} \tag{5.12}$$
$$\theta \sqcup (\theta \sqcap \theta') = \theta \tag{5.13}$$

Thereby, a number of conveniences mathematically and algorithmically are achieved as artefact models follow Boolean algebra.

Figure 5.9 shows how multiple inheritance of properties and object designators are lattice *meet* ($\sqcup$), and design moves in separate incomparable directions origins from lattice *join* ($\sqcap$) in a design lattice structure [13, 131]. In the figure, x_1 and x_2 are object designators and v_1, v_2, and g are property values. Furthermore, the figure illustrates the transitivity of property and object inheritance.

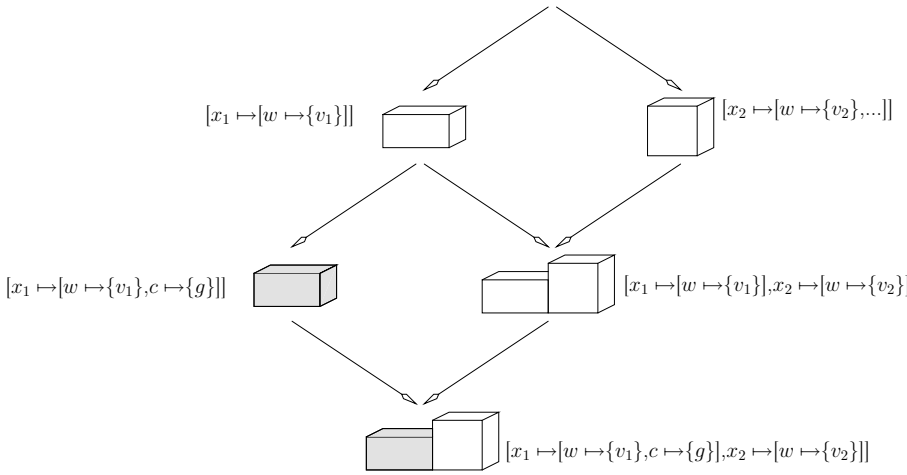


Figure 5.9: Design lattice.

A similar account holds for relations.

5.8.1 Design configuration and jump abstraction

We shall now explore some of the conveniences of design lattices namely in relation to what could be called *design jumps*.

By a design jump, we understand really performing a jump from one design configuration to another. Design jumps appear in situations where the practitioner shifts between two or more artefact models which appear in parallel. Such situations origins in the fact that the design process indeed follow different disciplines in which certain ideas may be discovered.

Certainly, the two configurations should exist in order for the jump to be possible. The two configurations are, however, not directly related by what we understand by a design move. They are configurations that exist in parallel to each other, and the common sets of objects, properties and relations may thus be quite small or even empty sets. Two artefact models appear in parallel in a design lattice, if they belong to different design aspect sub–lattices, at some higher level. E.g. an artefact model describing a yellow house may exist in parallel to an artefact model describing a red house because of the non-unifiability of the two colour values. Still, the practitioner may shift between these during a process of finding the right design configuration for meeting the requirements.

However, design jumps are orthogonal to any logical understanding of design. The reason is that we in principle can make a design jump from a model of a bridge to a model of a tunnel. Formally, we do not accept design jumps as design moves as it would undermine the partial order. The incremental principle is then lost.

In stead we aim at explaining design jumps by means of abstraction. An abstraction of a design jump is to backtrack the necessary steps in order to be able to perform design moves down the parallel path. This principle is shown in Figure 5.10.

The class partition approach would have difficulty handling abstractions of design jumps because steps upwards the structure may result in removal of objects, properties, or relations, not intended to be removed.

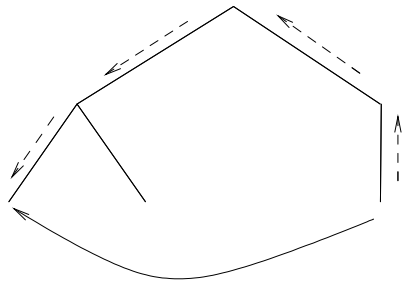


Figure 5.10: Abstraction of design jump.

5.9 Discussion

In this paper, we have proposed a formal foundation for incremental design. We have done so by suggesting that the design process can be recorded in a lattice structure where nodes correspond to design stages (artefact models) and edges correspond to the steps being made (design moves). We call such a lattice a *design lattice*. We have presented formal models in RSL of artefact models and the functions which change such models; the design moves.

Recording design processes as design lattices has a number of conveniences. Some of these, mentioned in this paper, include distinction between two sorts of design moves, eliminating unnecessary redundancy, and facilitating model composition in a mathematical way. Furthermore, it gives the opportunity to verify design steps by utilizing that an ordering relation holds between artefact models.

The specification of artefact models cuts down to a minimal set of ontological sorts: properties, binary relations, and values of these. In this framework the notion of object plays a minor rôle as the focus is on the characteristics of artefact. However, in order to avoid a problem of indiscernibility of property sets, identifiers of objects are introduced.

The presented understanding of the ontological entities necessary for representing designs as artefact models have founded the basis for formally specifying the two lattice operations join ($\sqcap$) and meet ($\sqcup$), as well as design moves for adding objects, properties of objects, and relations between objects. We have shown that design lattices indeed satisfy the criteria of being lattices. Thereby, design lattices have a number of mathematical properties, which eases implementation and management of design information. Also, the simplicity of the value and set–based representation of properties and relations makes design lattices suit-

able as foundation for future formal treatments of design like in definition of formal modelling languages and design algebras. The notion of artefact models is a simple one but have strong mathematical implication due to Boolean algebra being its foundation.

Further study may include the notion of decomposition of objects and an investigation of the relation between properties of object wholes and properties of the part of such wholes.

Finally, the notion of design lattices is similar to the notion of *refinement* in systematic software development, although the ordering symbol is syntactically reversed (see Section 2.3.2.5). It is similar in its motivation for recording design, but also in the possibility of utilising the partial ordering as a method for systematic designing. Therefore, future work may try to incorporate mathematical properties and concepts of refinement in the notion of design lattices.

Chapter 6

An algebraic specification of incremental, conceptual building design

Abstract: In this paper, we present a formal specification of incremental, conceptual building design. Our specification is written in The RAISE Specification Language (RSL) and is highly abstract and algebraic. Thus, representations of design stages are modelled as values of an abstract type, from which we can observe objects, properties of objects, relations between objects, and decompositions. We call such representations *artefact models*. Artefact models can now be changed and combined by a number of generator functions. These functions represent the ontologically fundamental design moves. We define a partial ordering on artefact models and show that the generator functions obey this ordering. Our contribution is thus to provide an alternative angle on the notion of incremental design as a process spanning what in Chapter 5 was called a *design lattice*.

6.1 Introduction

As a new foundation for software tools supporting incremental, conceptual design of buildings[1], we introduced the notion of *design lattices* in Chapter 5. In the paper we suggested a distinction between two sorts of design moves: Design moves *By aspect* and design moves *by configuration*. The former includes design moves of object conception, ascribing properties to objects, and adding relations between objects. The latter includes a design move for bringing partial[2] designs together. The two sorts of design moves, we claimed, originated from join and meet, respectively, in a lattice. Such a lattice is convenient for structuring design moves and can thus serve as foundation for design systems supporting incremental, conceptual design. A design lattice is thus introduced in order to facilitate structural representation of the design moves being made and the corresponding stages between the moves. Such a lattice structure may be represented in a data structure and capture performed design moves. Thereby, browsing the various design paths and combining different design stages. This is a necessary mechanism in distributed design where distinct practitioners work out individual partial design solutions. The approach, differs in particular from the approach in which possible design moves spans a tree structure due to class partition being its foundation. It does so by allowing partial designs from different paths to be combined. We wish to be able structure design information such that design moves maintains an ordering relation on designs stages. Thereby, redundancy is reduced and incrementality of the design process can be maintained. The latter is desirable as each design move should contribute alternately with knowledge of the artefact being designed. Lattices are convenient to accommodate these requirements and as argued in Chapter 5, design lattices may be a new angle to the discussion on what design is.

The partial relation maintains a principle of incrementality concerning objects, properties of objects, and relations. We believe that the concept of design lattices, in its mathematics, capture central issues in design in the case where design is brought into a formal context. Thus our contribution, we believe, is to clarify the concept of design when put in a formal framework motivated by potential software application.

In this paper, we explore the formal foundation for incremental, conceptual design further. However, we shall take a different approach as we define an algebraic specification of artefact models and design moves. By an algebraic speci-

[1] We here mention the concept of building design but in fact we could be concerned with the design of any kind of artefacts.

[2] Since all designs are abstract descriptions, they may all be considered partial. However, what we mean here is that two design stages are brought together to form a more complete (though not necessarily finish) design.

fication, we understand a formal specification which contains abstract datatypes (i.e. *sorts*) and functions on these types. The functions mainly fall in two categories: (i) Observer functions which state what information can be extracted from values of the abstract types, and (ii) generator functions which can change the internal state of these values.

Observer functions are the most abstract recognition rules we have as they can be used to model intractable domain concepts as well as technical solutions.

Also see [135] for further characteristics of algebraic specification methods. We use the RAISE Specification Language (RSL) as formal language [134, 135].

We shall again focus on the concept of design lattices. However, we aim at specifying each sort of design move (from both categories) instead of emphasizing the lattice operations as in Chapter 5. We redefine a partial ordering relation (derived from the one given in Chapter 5) on artefact models and prove that the presented design moves obey this ordering relation. That is, the design moves (modelled as generator functions) maintain incrementality in design processes.

6.1.1 The domain: Incremental design

The domain is that of conceptual design models and the operations which build up and elaborate on such models. The models we call *artefact models* as they are abstract descriptions of things to be man–made. The notion of artefact model is inspired by the notion of *artefaction* which is defined in [79]. The operations on artefact models are design moves. By a design move, we understand the cognitive or physical action of changing and elaborating a design representation. The notion of *move* in context of design originates in [149] and the notion has been picked up in various work [164, 139, 79, 62].

We follow [64, 67, 75, 167, 62] and understand design as an incremental process in which objects are introduced, properties are ascribed to objects, and objects are related.

It is convenient to define each design move as a move in the smallest sense. That is, to minimize the change of effect a design move has on the present design. Thereby, we are able to restrict to a minimum of fundamental generator functions for representing design moves. We thus, see all design moves as small steps.

6.1.2 Design lattices

As in Chapter 5, we consider the design moves of: conceiving an object, ascribing a property to an object, and adding a relation between two objects, to a model. Furthermore, we have the lattice operation *meet* which combines artefact models, and the lattice operation *join* which gives the artefact models having the intersection set of objects, properties, and relations.

In this paper we shall follow the principles of algebraic specification in RAISE [134, 135]. This means that each design move (modelled as a generator function) takes (among other parameters) a previous artefact model which is then modified. This principle is applied besides for the two lattice operations which apply on pairs of artefact models.

In addition to the design moves defined in Chapter 5, we introduce the design move of *decomposition*. This design move serves another cognitive purpose. Decomposition introduces a special relation holding between objects and sub–objects. However, the relation is not represented by edges in the design lattice as expected using mereological thinking. It is part of the artefact model itself.

The difference is that the part–whole relation ontologically is a more diffuse relation than a relation between an object and its properties, or binary (possibly topological) relations between objects. In Chapter 8 it is argued that the distinction between parts and whole often may be a matter of convention[3]. Often decomposition is used simply in order to be able to refer to an object as a whole and its parts. Other times decomposition is used as part–whole knowledge may come after the conception of the object being decomposed.

Many paths and many lattices can lead to the same design. Knowledge of an artefact may be conceived and added to the model in different orders. Also, similar designs may be achieved by taking either a bottom up or top–down approach. Figure 6.1 shows a bottom up approach in which three objects are added in parallel paths[4]. The objects are independently specialised by adding properties, and are finally put together to form an entrance section for a house.

Figure 6.2 shows a top–down approach which leads to a similar design[5]. In this approach a single object is introduced, specialised, and then decomposed into three sub–objects forming a similar entrance section.

[3] Socially or based on the present use of language.

[4] We use dashed lines in lack of better graphical ways of depicting object without properties (and thus also without specific dimension and size.)

[5] We use dashed boxes with round edges in order not to confuse lines representing part–whole relations and arrows representing design moves.

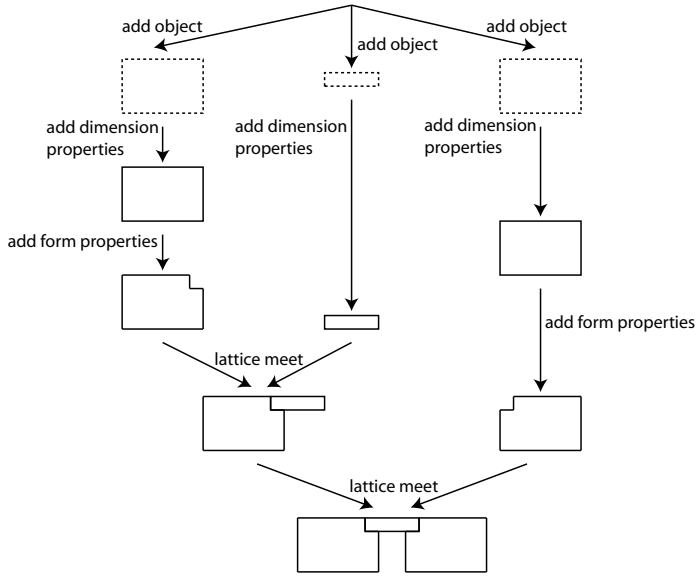


Figure 6.1: Bottom–up design lattice.

6.2 Artefact models

In the following, we present an algebraic specification of artefact models (θ:Θ) containing: objects (x:X), properties (p:P) of objects, and relations (r:R) between objects. A property is considered a pair of which the first component is called the *attribute*[6] (a:AP) and the latter is a set of values[7] (vs:VP-**infset**). A set of values may be infinite in order to express properties like weighing at least 200 pounds. An attribute uniquely identifies a set of values for a given object. An example of an attribute is *colour*. The corresponding set of property values could be $\{blue, grey, green\}$. A set with more than one element means that the object can be realised in more than one way; namely one for each of the property values.

If an object has properties with attributes $a1, \ldots, am$ and corresponding values $\{v1_1, \ldots, v1_{n1}\}, \ldots, \{vm_1, \ldots, v1_{nm}\}$, the number of possible realisations is $n1 \times \ldots \times nm$.

The empty set of values corresponds to absurdum: no possible realisation exists. Absurdum can be used to designate an error which may occur because of

[6] or property attribute.
[7] called property values.

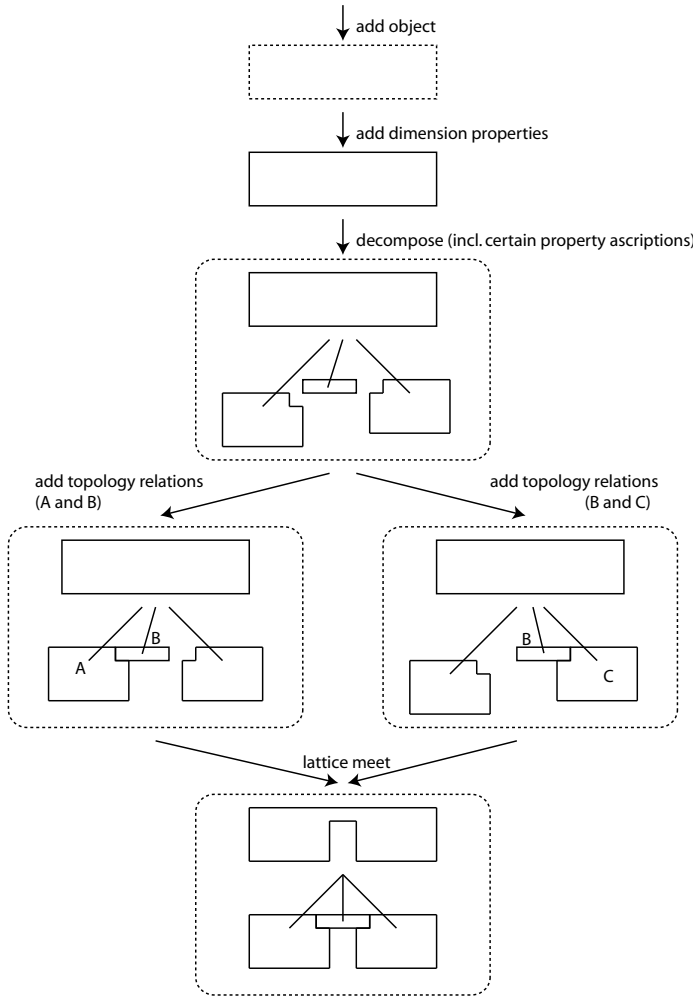


Figure 6.2: Top–down design lattice.

conflicting models on which lattice meet is applied. A similar account is given for relations. A relation between two objects is a pair of two component. The first is the attribute[8] (a:AR) and the second is a set of values[9] (vs:VR-**infset**). An example of an attribute of a relation is *horizontal–distance* for which the corresponding relation values could be $\{20m, 30m, 60m\}$. Another example of an attribute is *placement* with the values $\{on\_top, next\_to\}$. As it appears, many relations concern topology.

[8] or relation attribute.
[9] called relation values.

6.2.1 Observer functions

From an artefact model (θ:Θ), we can observe the objects (xs:X-**set**) which makes the model. Given an object in a model, we can observe the properties (ps:P-**set**) ascribed to that object in the model. From an artefact model, we can observe the relations (rs:R-**set**) added to that model. These relations are given explicitly, including the object pairs of concern. Given an object in a model, we can observe the set of sub–objects into which the argument object has been decomposed. In case the object has not been decomposed, this set is empty. Properties are attribute–value pairs ((a,vs):(AP×VP)) as are relations ((a,vs):(AR×VR)). Note, that we distinguish between attributes of properties, and attributes of relations, as well as between values of properties and relations.

type
  Θ,
  X,
  P = AP × VP-**infset**,
  R = AR × VR-**infset**,
  AP, AR,
  VP, VR

value
  [observer functions]
  objects: Θ → X-**set**,
  properties: X × Θ → P-**set**,
  relations: Θ → ((X × X) × R)-**set**,
  decomposition: X × Θ → X-**set**,

6.2.2 Concerning properties in decomposition

Before we can state a consistency axiom for artefact models, we need an answer to the question of whether properties are to be considered in this context. That is, do the properties of a decomposed object relate to the properties of the sub–objects? The following discussion aims at reaching a clarification in answering this question. We conclude that no such relation can be claimed. Our discussion goes as follows (the section can easily be skipped if interest is not on the ontological or epistemological aspects):

We consider the problem of how to understand the relation between properties possessed by an object and the properties which seem to be derived to

sub–objects in a decomposition. We name this relation the *derivation relation* assuming that derivation concerns properties. For convenience we define the notion of *attribution* as follows: An object is said to be attributed with an attribute a if the object is ascribed a property $p = \{a, vs\}$; where vs is some property value set.

Our quest here is problematic though our solution is uncontroversial. The quest is problematic in the sense that the derivation relation in question seems not to be in focus either in literature on the metaphysics of properties, neither in literature on part–whole theories. Though, Simon treats a similar question in context of essential parts and defines the notion of *local predication* (see page 135 in [155]). Also, Simons states that predicates over objects as sums (the objects we decompose) are to be *cumulative* (see page 111 in [155]). If a predicate applies to a part, it must apply to the whole as well. But according to Simons, Quine has argued the absurdity in this perspective: It simply does not hold for *mass*–predicates. However, if we claim a relation to hold — as a conjecture — we should at least try out some different approaches as we intend to do here.

To start with, we shall dogmatically assume that we need to say *something* about that derivation relation. That is, there must indeed be a wellformed condition for properties of objects and properties of sub–objects.

A first approach could be that we allow decompositions in which sub–objects not necessarily derive all attributes from the properties of the object being decomposed. The motivation here is that we often may speak of an object having a property even though we know that the object consists of parts which do not possess the property. As an example, consider a table. We may ascribe the property of having the colour blue to that table even though the legs have the colour grey. Thus, motivation comes from our (perhaps vague) way of using natural language. It is one of the ways we perform abstractions in every day life: "*This chair is blue*" may be true in our conception even though only main parts are blue. Let us consider it formally and in our context. The approach means that the set of attributes to be derived is a subset of the attributes which are attributed the object being decomposed.

The approach emphasizes the situation where this subset is a *proper* subset. The situation, however, opens for the possibility of having the empty set of as a special case. If the derived set of attributes is empty, there is no relation to be claimed. It turns out in our formalisation that this implies that there are not sub–objects then. However, this was not our intention, and if we try to accommodate by choosing another formalisation, we discover that the problem is much more far reaching. If a certain attribute of a decomposed object is not designated as to be derived, we allow a property with this attribute to be ascribed the sub–objects after the decomposition. This latter property could

have a different value set, but more problematic; the attribute could be different, so that the property belongs to a different ontological domain of properties. But this is absurd. A wall which is ascribed the property of being made of concrete can then be decomposed into two parts which all are later ascribed the property of being made of wood. That is, we cannot say anything about the derivation relation. In defence, we shall allow for partial decomposition. It may be that a property, which is not derived in one lattice path, is derived in another. Still, we have not said anything about the derivation relation; only that some sort of consistency should exist..

A second approach is to try to repair the leaks of the first one. Leaving out an attribute of a decomposed object may lead to absurdities. We might require that at least one sub–object derives a property with that attribute. That is, in order for the decomposition to be wellformed with respect to derived properties, the derivation relation should maintain that all properties are derived to sub–objects in some way. We can divide this approach into two cases: (i) We could require that at least one sub–object derives all the properties of the decomposed object, or (ii) we could require that all properties are at least derived in some way to some sub–object; i.e. distributed. The approach, in either case, introduces the notion of *essential parts*. In (i) the object which derives all properties, is considered essential and thus more important than the other sub–objects. In (ii) such importance depends on what perspective we take when "considering" the decomposed object. In one perspective, material may be essential; in another, mass may be essential, etc. The difference between (i) and (ii) is thus a matter of context, and we shall treat them together. The second approach (although quite appealing) has a built in problem which is rooted in distinguishing essential from non–essential parts. Take for example a television. We say that this television is black even though some parts of it do not possess this property. Let us now decompose the television object into a black box, electronics inside, glass front and a little red lamp flashing. The essential part, concerning the property mentioned, may be the black box; perhaps because it spatially dominates the outer surfaces of the television. However, for that we cannot be sure. It could as well be the glass front which is dominating. But then we might also say that the red light is the essential part, and so on. If we wish to select the second approach, we should at least have a formal way of distinguishing essential and non–essential parts. This issue is in fact a known problem in mereology and we are here far from a real clarification [155]. Furthermore, since we allowed for partial decomposition it may be that no sub–objects can be designated as essential at the time of decomposition.

A third approach is to require that what is said about an object should also apply for all sub–objects into which it is decomposed. We thus avoid the tricky ways of using language, which were mentioned for the first approach. In decomposing an object, we should make sure to designate all attributes of the decomposed

object as to apply to all sub–objects as well; and similar for value sets. For colour properties and material properties, the approach seems appealing. If a decomposed object is red, its parts should be too, and if a decomposed object is made of wood so should its parts. The trouble appears when considering properties like dimensions. If a beam has a length of $2m$ any proper part of it certainly does not.

A fourth approach is to repair the lacks of the third approach. We maintain the requirement that all attributes of a decomposed object are attributed sub–objects as well. However, we loosen the restriction on value sets. The derivation relation thus only says something about the kind of properties, not what specific properties. However, this is hardly the way a design practitioner works. If we are not allowed to state — using a top–down approach — that a house is to be made of wood because the chimney is not, we loose the fuzzy way designers abstract. The abstraction process may here be impossible to define explicitly and universally.

A fifth variant could be considered as solution to the problem of the fourth approach. It is as appealing as it is strange, which is partly why it has not been chosen in this paper. Still, if the notion of abstraction — as considered under the fourth approach — is to be defined explicitly, this fifth variant we believe is the best candidate for a foundation. Consider an object with a property p. The object is now decomposed into a number of sub–objects. The rule is now that if a sub–object derives a property, this property does not apply to the decomposed object anymore, if not all sub–objects derive the property. That is, it is removed.

We conclude that there seems not to be any derivation principle which satisfies both the requirement of stating a consistent, universal rule for derivation of properties, and at the same time do not restrict the design process. The seemingly strange solution of the fifth variant needs more clarifications before it can be called a real candidate. Therefore, we have chosen not to claim there to be a relation between properties of an object and properties of the objects into which it is decomposed.

6.2.3 Consistency

In order for an artefact model to be consistent, it must satisfy the following: (i) All attributes for properties of an object must uniquely identify a corresponding value set, and (ii) all attributes for relations must uniquely identify a corresponding value set.

[consistency]
axiom ∀ θ:Θ, x,x′:X, a:AP, vs, vs′:VP**-infset**, a′,AR, vs″,vs‴:VR**-infset** •
  ((a,vs) ∈ properties(x,θ) ∧ (a,vs′) ∈ properties(x,θ) ⇒ vs = vs′) ∧
  (((x,x′),(a′,vs″)) ∈ relations(θ) ∧ ((x,x′),(a′,vs‴)) ∈ relations(θ) ⇒
    vs″=vs‴)

The algebraic laws of Section 6.4 ensure non–cyclicity of decompositions. Sub–objects which are introduced by decompositions, are new objects and cannot be objects already existing in the model.

6.3 Design moves

We shall follow the distinction between design moves by aspect and design moves by configuration, as defined in Chapter 5.

6.3.1 Design move by aspect

Design moves of the first class are: (i) Adding an object to a model, (ii) ascribing a property to an object in a model, and (iii) adding a relation between two objects to a model. These are common in the sense that their application, contribute with knowledge of the artefact in mind. We see this "adding of knowledge" as restriction on the scope of possible interpretations of the model into reality. Thus, we shall talk of specialisation. Adding an object makes the design more specialised as we then require the existence of such an object. Adding a property to an object makes the design more specialised as we then require the object to possess that property. Adding a relation between two objects makes the design more specialised as we then require these objects to be related in this way.

Each of the described design moves is modelled as a generator function:

value
  add_object: X × Θ $\xrightarrow{\sim}$ Θ,
  add_property: P × X × Θ $\xrightarrow{\sim}$ Θ,
  add_relation: R × (X × X) × Θ $\xrightarrow{\sim}$ Θ,

All three generator functions are partial. Adding an object to a model in which

that object already exists yields problems in determining what properties that resulting object should have. It seems most reasonable that observing the properties of a newly added object gives the empty set. For adding an object, it is thus required that the object is not already present in the model. On the other hand, ascribing a property to an object requires existence of that object in the argument model. For adding relations, we require that both objects are present in the argument model. We thus define the following guards:

value
[guards]
is_object_present: X × Θ → **Bool**
is_object_present(x,θ) ≡ x ∈ objects(θ),

is_property_attribute_present: AP × X × Θ → **Bool**
is_property_attribute_present(a,x,θ) ≡
(∃ vs:VP-**infset** • (a,vs) ∈ properties(x,θ)),

is_relation_attribute_present: AR × (X × X) × Θ → **Bool**
is_relation_attribute_present(a,(x,x′),θ) ≡
(∃ vs:VR-**infset** • ((x,x′),(a,vs)) ∈ relations(θ)),

6.3.2 Design move by configuration: Lattice operations

The two lattice operations are generator function which apply to pairs of artefact models:

value
meet: Θ × Θ → Θ,
join: Θ × Θ → Θ

It is convenient, explicitly to overload similar operations for sets of properties and relations. For relations, we need to include pairs of related objects. We have:

value
meet: P-**set** × P-**set** → P-**set**,
join: P-**set** × P-**set** → P-**set**,

meet: ((X×X) × R)-**set** × ((X×X) × R)-**set** → ((X×X) × R)-**set**,
join: ((X×X) × R)-**set** × ((X×X) × R)-**set** → ((X×X) × R)-**set**

axiom
[property set axioms]
∀ ps, ps′:P-**set** •
meet(ps,ps′) ≡
{(a,vs)|(a,vs):AP×VP-**infset** •
(∃ vs′,vs″:VP-**infset** •
((a,vs′) ∈ ps ∧ (a,vs″) ∈ ps′ ∧ vs=vs′ ∩ vs″) ∨
((a,vs′) ∈ ps ∧ (a,vs″) ∉ ps′ ∧ vs=vs′) ∨
((a,vs′) ∉ ps ∧ (a,vs″) ∈ ps′ ∧ vs=vs′))},

∀ ps, ps′:P-**set** •
join(ps,ps′) ≡ {(a,vs)|(a,vs):AP×VP-**infset** •
(∃ vs′,vs″:VP-**infset** • (a,vs′) ∈ ps ∧ (a,vs″) ∈ ps′ ∧ vs=vs′ ∪ vs″)},

[relation set axioms]
∀ rs, rs′:R-**set** •
meet(rs,rs′) ≡
{((x,x′),(a,vs))|(X×X)×P •
(∃ vs′,vs″:VR-**infset** •
(((x,x′),(a,vs′)) ∈ rs ∧ ((x,x′),(a,vs″)) ∈ rs′ ∧ vs=vs′ ∩ vs″) ∨
(((x,x′),(a,vs′)) ∈ rs ∧ ((x,x′),(a,vs″)) ∉ rs′ ∧ vs=vs′) ∨
(((x,x′),(a,vs′)) ∉ rs ∧ ((x,x′),(a,vs″)) ∈ rs′ ∧ vs=vs″))},

∀ rs, rs′:R-**set** •
join(rs,rs′) ≡ {((x,x′),(a,vs))|(X×X)×P •
(∃ vs′,vs″:VR-**infset** •
((x,x′),(a,vs′)) ∈ rs ∧ ((x,x′),(a,vs″)) ∈ rs′ ∧ vs=vs′ ∪ vs″)}

We see that lattice meet applied on two property sets (ps:P-**set**) and (ps′:P-**set**), gives the property set which is the union set of property attributes from *ps* and *ps′*. For common property attributes of *ps* and *ps′* the set of property values is the intersection set of values taken from *ps* and *ps′*. A similar account applies to relations. Lattice join applied on two property sets (ps:P-**set**) and (ps′:P-**set**) is the property set which is the common property attributes of *ps* and *ps″*. For common property attributes of *ps* and *ps′*, the set of property values is the union set of values from *ps* and *ps′*.

6.3.3 The design move of decomposition

In addition to the three design moves described in Section 6.3.1, we introduce a generator function for the design move of decomposition. The reason for including this design move is to accommodate top–down designing where artefact models are specialised by decomposing objects into sub–objects. Such a design move contributes with knowledge of an objects parts.

The design move of decomposition is declared as the generator function:

value
decompose: X × (AP $\xrightarrow[m]{}$ (X $\xrightarrow[m]{}$ VP-**infset**)) × Θ $\xrightarrow{\sim}$ Θ

The second argument defines a kind of rule for the decomposition. For each property attribute (a:AP) of the argument object, we get a map from the new sub–objects to their individual value sets. The rule is a way of specifying how the sub–objects derive properties from the object which is decomposed. However, it is only a convenient way of comprising a sequence of specialisation operations. To make this clear, we introduce a number of terminological definitions:

Definition 6.1 An object x_p is called a *parent object* of another object x_s if x_p has been decomposed into a set of objects including x_s. x_s is then called a *sub–object* of x_p.

A sub–object may get a number of attributes from properties of its parent object. The reason is that knowledge of an object may also say something about the parts of which that object consists. Such knowledge is expressed by the decomposition rule.

Definition 6.2 An object x is said to *derive* a property (a, vs) from another object x' if x is a sub–object of x' and a is an attribute of x as well as of x'.

If a table is described as possessing the property of being made of wood this property contains an attribute *material*. The property is also possessed by the various parts of the table. However, it may be the case that only the attribute is the same for sub–object and parent object. E.g. for the property of having a certain length. Decomposing a beam into parts conceptually ascribes different length values to the sub–objects, though the attribute "length" is maintained (see Section 6.3.3). However, as argued in Section 6.2.2, we cannot in general state a relation between the properties of a sub–object and the properties of

a parent–object. The decomposition rule is therefore to be understood in a constructive sense.

Three issues are important in context of decomposition rules. First: we allow that objects are not decomposed at once into all their parts. We call this *partial decomposition*. Second: Although we allow for partial decompositions, we do not allow for an object to be decomposed several times in distinct operations (if following the same path in the design lattice). This may seem strange, however, the argumentation is similar to the one for not allowing an object to be ascribed a property with the same attribute twice, following the same path in the design lattice. We want dogmatically to maintain a principle of incrementality. If an object needs to be decomposed in two steps we see these as alternative decompositions of the same object, belonging to parallel paths of the design lattice. The two decomposition configurations can then be combined using lattice meet as shown on Figure 6.3. Stepwise decomposition can thus be modelled as applying lattice meet on alternative (partial) decompositions.

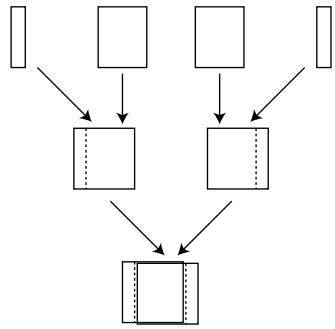


Figure 6.3: Stepwise decomposition using lattice meet.

As a special case of this issue, we allow for sub–objects to "overlap" spatially. The reason for this is that it can be convenient to talk about the same object in context of its decomposition relation to different parent objects (see Figure 6.3).

Third, we have an issue concerning what happens if an already decomposed object is ascribed properties. For the parent object the situation is similar to when an object is ascribed a property with add_property. For the sub–objects, however, the status maintains the same as we do not require a relation between properties of parent objects and properties of sub–objects.

A decomposition is valid if and only if the argument object is in the argument model, the argument object is not designated as a sub–object, and each property attribute in the rule is an attribute of a property which is ascribed the argument object.

is_valid_decomp: X × (AP $\overrightarrow{m}$ (X $\overrightarrow{m}$ VP-**infset**)) × Θ → **Bool**
is_valid_decomp(x,am,θ) ≡
  x ∈ objects(θ) ∧
  (∀ a:AP • a ∈ **dom** am ⇒ x ∉ **dom** am(a)) ∧
  (∀ a:AP • a ∈ **dom** am ⇒ (∃ vs:VP-**infset** • (a,vs) ∈ properties(x,θ)))

In addition to the consistency axiom defined in Section 6.2.3, we require that if an object is a sub–object in a decomposition, it cannot be an object in the previous artefact model:

value
  [decomp−consistency]
  ∀ x, x′:X, am:AP $\overrightarrow{m}$ (X $\overrightarrow{m}$ VP-**infset**), θ:Θ •
    x ∈ decomposition(x′,decompose(x′,am,θ)) ∧
    (∃ a:AP • a ∈ **dom** am ∧ x ∈ **dom** am(a)) ⇒ x ∉ objects(θ)

When defining decomposition algebraically, we need a function which given a decomposition rule and a sub–object, gives the properties that the sub–object derives according to the rule.

value
  [misc]
  derive: AP $\overrightarrow{m}$ (X $\overrightarrow{m}$ VP-**infset**) × X → P-**set**
  derive(am,x) ≡
    {(a,vs)|(a,vs):P • a ∈ **dom** am ∧ x ∈ am(a) ∧ vs=am(a)(x)}

6.4 Algebraic axioms

We now specify a number of axioms which aims at relating observer functions to generator functions.

6.4.1 Observing objects

axiom
  [objects]

∀ θ:Θ, x:X •
  objects(add_object(x,θ)) ≡ objects(θ) ∪ {x},
pre ∼is_object_present(x,θ),

∀ θ:Θ, (a,vs):P, x:X •
  objects(add_property((a,vs),x,θ)) ≡ objects(θ)
pre is_object_present(x,θ) ∧ ∼is_property_attribute_present(a,x,θ),

∀ θ:Θ, (a,vs):R, x, x′:X •
  objects(add_relation((a,vs),(x,x′),θ)) ≡ objects(θ)
pre is_object_present(x,θ) ∧ is_object_present(x′,θ) ∧
    ∼is_relation_attribute_present(a,(x,x′),θ),

∀ θ, θ′:Θ •
  objects(meet(θ,θ′)) ≡ objects(θ) ∪ objects(θ′),

∀ θ, θ′:Θ •
  objects(join(θ,θ′)) ≡ objects(θ) ∩ objects(θ′),

∀ θ:Θ, x:X, am:AP $\xrightarrow[m]{}$ (X $\xrightarrow[m]{}$ VP-**infset**) •
  objects(decompose(x,am,θ)) ≡ objects(θ) ∪
    {x′|x′:X • (∃ a:AP • a ∈ **dom** am ∧ x′ ∈ **dom** am(a))}
pre is_valid_decomp(x,am,θ),

6.4.2 Observing properties

axiom
  [properties]
  ∀ θ:Θ, x,x′:X •
    properties(x,add_object(x′,θ)) ≡
      if x=x′ **then** {} **else** properties(x,θ) **end**
  pre ∼is_object_present(x′,θ),

  ∀ θ:Θ, (a,vs):P, x,x′:X •
    properties(x,add_property((a,vs),x′,θ)) ≡
      if x=x′ **then** properties(x,θ) ∪ {(a,vs)}
      else properties(x,θ)
      end
  pre is_object_present(x′,θ) ∧ ∼is_property_attribute_present(a,x′,θ),

  ∀ θ:Θ, (a,vs):R, x,x′,x″:X •

properties(x,add_relation((a,vs),(x′,x″),θ)) ≡ properties(x,θ)
pre is_object_present(x′,θ) ∧ is_object_present(x″,θ) ∧
∼ is_relation_attribute_present(a,(x′,x″),θ),

∀ θ, θ′:Θ, x:X •
properties(x,meet(θ,θ′)) ≡ meet(properties(x,θ),properties(x,θ′)),

∀ θ, θ′:Θ, x:X •
properties(x,join(θ,θ′)) ≡ join(properties(x,θ),properties(x,θ′)),

∀ θ:Θ, x,x′:X, am:AP $\xrightarrow[m]{}$ (X $\xrightarrow[m]{}$ VP-**infset**) •
properties(x,decompose(x′,am,θ)) ≡
if x ∈ decomposition(x′,θ) **then**
derive(am,x)
else properties(x,θ)
end
pre is_valid_decomp(x,am,θ)

6.4.3 Observing relations

axiom
[relations]
∀ θ:Θ, x:X •
relations(add_object(x,θ)) ≡ relations(θ)
pre ∼is_object_present(x,θ)

∀ θ:Θ, (a,vs):P, x:X •
relations(add_property((a,vs),x,θ)) ≡ relations(θ),
pre is_object_present(x,θ) ∧ ∼is_property_attribute_present(a,x,θ),

∀ θ:Θ, (a,vs):R, x, x′:X •
relations(add_relation((a,vs),(x,x′),θ)) ≡ relations(θ) ∪ {((x,x′),(a,vs))}
pre is_object_present(x,θ) ∧ is_object_present(x′,θ) ∧
∼is_relation_attribute_present(a,(x,x′),θ),

∀ θ,θ′:Θ •
relations(meet(θ,θ′)) ≡ meet(relations(θ),relations(θ′)),

∀ θ,θ′:Θ •
relations(join(θ,θ′)) ≡ join(relations(θ),relations(θ′)),

∀ θ:Θ, x:X, am:AP $\xrightarrow[m]{}$ (X $\xrightarrow[m]{}$ VP-**infset**) •
  relations(decompose(x,am,θ)) ≡ relations(θ)
pre is_valid_decomp(x,am,θ)

6.4.4 Observing decompositions

axiom
[decomposition]
∀ θ:Θ, x,x′:X •
  decomposition(x,add_objects(x′,θ)) ≡
    if x=x′ **then** {} **else** decomposition(x,θ) **end**
pre ∼is_object_present(x′,θ),

∀ θ:Θ, x,x′:X, (a,vs):P •
  decomposition(x,add_property((a,vs),x′,θ)) ≡ decomposition(x,θ)
pre is_object_present(x′,θ) ∧ ∼is_attribute_present(a,x′,θ)

∀ θ:Θ, x, x′,x″:X, (a,vs):R •
  decomposition(x,add_relation((a,vs),(x′,x″),θ)) ≡ decomposition(x,θ)
pre is_object_present(x′,θ) ∧ is_object_present(x″,θ) ∧
    ∼is_relation_attribute_present(a,(x′,x″),θ),

∀ θ, θ′:Θ, x:X •
  decomposition(x,meet(θ,θ′)) ≡
    decomposition(x,θ) ∪ decomposition(x,θ′),

∀ θ, θ′:Θ, x:X •
  decomposition(x,join(θ,θ′)) ≡
    decomposition(x,θ) ∩ decomposition(x,θ′),

∀ θ:Θ, x,x′:X, am:AP $\xrightarrow[m]{}$ (X $\xrightarrow[m]{}$ VP-**infset**)
  decomposition(x,decompose(x′,am,θ)) ≡
    if x=x′ **then**
      {x″|x″:X • (∃ a:AP • a ∈ **dom** am ∧ x″ ∈ **dom** am(a))}
    elsif x ∈ {x″|x″:X •
      (∃ a:AP • a ∈ **dom** am ∧ x″ ∈ **dom** am(a))} **then** {}
    else decomposition(x,θ)
    end
pre is_valid_decomp(x′,am,θ)

6.5 Partial order

In Chapter 5, the notion of artefact models was defined by two mappings *objects* and *relations*. In this paper, the job of these are performed by observer functions. In addition, we have added the notion of decomposition.

As mentioned in Section 6.1, maintaining some sort of ordering relation between designs is necessary in order to assure that incrementality is not violated. In order to show that the specification of artefact models and the associated generator functions defines a lattice, we need to show that each design move does not violate the partial order relation between artefact models; non–ambiguity for lattice meet and join has been proved in Chapter 5.

We shall approach this problem by rewriting the definition of the ordering relation $(\leq, \Theta)$ into a version which complies with the specification of artefact models given in this paper. By $\theta' \leq \theta$ we mean that the artefact model θ' is more specialised than the model θ. We use the symbol $\leq$ to emphasize that design moves are non–increasing on value sets. A design move from a to b thus should preserve the ordering $b \leq a$.

In order to rewrite the axiom for partial order of artefact models presented in Chapter 5, we first do the following observation. The domains of the mappings *objects* and *relations* in the axiom from Chapter 5 correspond to the observer functions *objects* and *relations* in this paper.

We can thus formulate the ordering relation $(\leq, \Theta)$ as:

value
  $\leq$: $\Theta \times \Theta \rightarrow$ **Bool**

axiom $\forall$ θ, θ':Θ •
  $\theta' \leq \theta \equiv$
    ($\forall$ x:X • x $\in$ objects(θ) $\Rightarrow$ (x $\in$ objects(θ') $\wedge$
      ($\forall$ a:AP, vs:VP-**infset** • (a,vs) $\in$ properties(x,θ) $\Rightarrow$
        ($\exists$ vs′:VP-**infset** • (a,vs′) $\in$ properties(x,θ') $\wedge$ vs′ $\subseteq$ vs′)))) $\wedge$
    ($\forall$ (x,x′):(X$\times$X), a:AR, vs:VR-**infset** •
      ((x,x′),(a,vs)) $\in$ relations(θ) $\Rightarrow$
        ($\exists$ vs′:VR-**infset** • ((x,x′),a,vs′) $\in$ relations(θ') $\wedge$ vs′ $\subseteq$ vs)) $\wedge$
    ($\forall$ x:X • decomposition(x,θ) $\subseteq$ decomposition(x,θ'))

We notice that the axiom can be split up into the following parts (assuming a conjunction of the parts):

$(\forall\ \theta,\theta'{:}\Theta,\ x{:}X \bullet x \in objects(\theta) \Rightarrow x \in objects(\theta'))$,

$(\forall\ \theta,\theta'{:}\Theta,\ x{:}X,\ a{:}AP,\ vs'{:}VP\textbf{-infset} \bullet$
$\quad x \in objects(\theta) \Rightarrow (a,vs) \in properties(x,\theta) \Rightarrow$
$\quad\quad (\exists\ vs'{:}VP\textbf{-infset} \bullet (a,vs') \in properties(x,\theta') \wedge vs' \subseteq vs))$,

$(\forall\ \theta,\theta'{:}\Theta,\ (x,x'){:}(X\times X),\ a{:}AR,\ vs,\ vs'{:}VR\textbf{-infset} \bullet$
$\quad ((x,x'),(a,vs)) \in relations(\theta) \Rightarrow$
$\quad\quad (\exists\ vs'{:}VR\textbf{-infset} \bullet ((x,x'),a,vs') \in relations(\theta') \wedge vs' \subseteq vs))$,

$(\forall\ \theta,\theta'{:}\Theta,\ x{:}X \bullet decompositions(x,\theta) \subseteq decompositions(x,\theta'))$,

We shall take advantage of this paraphrase in the proofs.

The unambiguous existence of least upper bound and greatest lower bound have already been proven in Chapter 5 for a definition of partial order which includes requirements besides those to decomposition. However, proving this for decomposition is trivial as it follows Boolean algebra. Also, changes to the definition of lattice meet and lattice join have not changed significantly from those given in Chapter 5. Thus, we shall not repeat the proves of these properties here. Left is to prove that the operations add_object, add_property, add_relation, decompose, union, and join, do not violate the partial order of artefact models.

Proposition 6.3 *The design move given by the generator function* ***add_object*** *order preserves* $(\leq,\Theta)$.

Proof: We show that the generator function do not violate the axiom for the partial order.

$\forall\ \theta,\theta',\Theta,\ x{:}X \bullet x \in objects(\theta) \Rightarrow x \in objects(\theta')$

θ'=add_object(x′,θ):

$x{:}X \bullet x \in objects(\theta) \Rightarrow x \in objects(add\_object(x',\theta))$

$\equiv\ x{:}X \bullet x \in objects(\theta) \Rightarrow x \in (objects(\theta) \cup \{x'\})$

$\equiv$ **true**

$\forall\ \theta,\theta'{:}\Theta,\ x{:}X,\ a{:}AP,\ vs{:}VP\textbf{-infset} \bullet$
$(x \in objects(\theta) \wedge (a,vs) \in properties(x,\theta)) \Rightarrow$
$(\exists\ vs'{:}VP\textbf{-infset} \bullet (a,vs') \in properties(x,\theta') \wedge vs' \subseteq vs)$

θ'=add_object(x′,θ):

$(x \in objects(\theta) \wedge (a,vs) \in properties(x,\theta)) \Rightarrow$
$(\exists\ vs'{:}VP\textbf{-infset} \bullet (a,vs') \in properties(x,add\_object(x',\theta)) \wedge$
$vs' \subseteq vs)$

x=x′:

$(x \in objects(\theta) \wedge (a,vs) \in properties(x,\theta)) \Rightarrow$
$(\exists\ vs'{:}VP\textbf{-infset} \bullet (a,vs') \in \{\} \wedge vs' \subseteq vs)$

From pre–condition ∼is_object_present(x′,θ) we have that x$\notin$objects(θ).

$\equiv$ **true**

x≠x′:

$(x \in objects(\theta) \wedge (a,vs) \in properties(x,\theta)) \Rightarrow$
$(\exists\ vs'{:}VP\textbf{-infset} \bullet (a,vs') \in properties(x,\theta) \wedge vs' \subseteq vs)$

From axiom [consistency] we have that vs=vs′.

$\equiv$ **true**

$\forall\ (x,x'){:}(X\times X),\ a{:}AR,\ vs{:}VR\textbf{-infset},\ \theta,\theta'{:}\Theta \bullet$
$((x,x'),(a,vs)) \in relations(\theta) \Rightarrow$
$(\exists\ vs'{:}VR\textbf{-infset} \bullet ((x,x'),(a,vs')) \in relations(\theta') \wedge vs' \subseteq vs)$

θ'=add_object(x′,θ):

$((x,x'),(a,vs)) \in \text{relations}(\theta) \Rightarrow$
$\quad (\exists\ vs':VR\textbf{-infset} \bullet ((x,x'),(a,vs')) \in \text{relations}(\text{add\_object}(x',\theta))$
$\qquad \wedge\ vs' \subseteq vs)$

$\equiv ((x,x'),(a,vs)) \in \text{relations}(\theta) \Rightarrow$
$\qquad (\exists\ vs':VR\textbf{-infset} \bullet ((x,x'),(a,vs')) \in \text{relations}(\theta) \wedge vs' \subseteq vs)$

From axiom [consistency] we have that $vs=vs'$.

$\equiv$ **true**

$\forall\ \theta,\theta':\Theta,\ x:X \bullet \text{decomposition}(x,\theta) \subseteq \text{decomposition}(x,\theta')$

$\underline{\theta'=\text{add\_object}(x',\theta)}$:

$\text{decomposition}(x,\theta) \subseteq \text{decomposition}(x,\text{add\_object}(x',\theta))$

$\underline{x=x'}$:

$\text{decomposition}(x,\theta) \subseteq \{\}$

From pre–condition $\sim\text{is\_object\_present}(x',\theta)$ we have:

$\equiv \{\} \subseteq \{\}$

$\equiv$ **true**

$\underline{x\neq x'}$:

$\text{decomposition}(x,\theta) \subseteq \text{decomposition}(x,\theta)$

$\equiv$ **true**

QED

□

Proposition 6.4 *The design move given by the generator function* ***add_property*** *order preserves* $(\leq,\Theta)$.

Proof: We show that the generator function do not violate the axiom for the partial order.

$\forall$ x:X • x $\in$ objects(θ) $\Rightarrow$ x $\in$ objects(θ')

θ'=add_property((a,vs),x′,θ):

x:X • x $\in$ objects(θ) $\Rightarrow$ x $\in$ objects(add_property((a,vs),x′,θ))

$\equiv$ x:X • x $\in$ objects(θ) $\Rightarrow$ x $\in$ objects(θ)

$\equiv$ **true**

$\forall$ θ,θ:Θ, x:X, a:AP, vs:VP-**infset** •
(x $\in$ objects(θ) $\wedge$ (a,vs) $\in$ properties(x,θ)) $\Rightarrow$
($\exists$ vs′:VP-**infset** • (a,vs′) $\in$ properties(x,θ')) $\wedge$ vs′ $\subseteq$ vs

θ'=add_property((a″,vs″),x′,θ):

(x $\in$ objects(θ) $\wedge$ (a,vs) $\in$ properties(x,θ)) $\Rightarrow$
($\exists$ vs′:VP-**infset** • (a,vs′) $\in$ properties(x,add_property((a″,vs″),x′,θ)) $\wedge$
vs′ $\subseteq$ vs)

x=x′:

(x $\in$ objects(θ) $\wedge$ (a,vs) $\in$ properties(x,θ)) $\Rightarrow$
($\exists$ vs′:VP-**infset** • (a,vs) $\in$ (properties(x,θ) $\cup$ {(a″,vs″)}) $\wedge$
vs′ $\subseteq$ vs)

From axiom [consistency] we have that vs=vs′.

$\equiv$ **true**

else:

$\equiv$ $(x \in objects(\theta) \wedge (a,vs) \in properties(x,\theta)) \Rightarrow$
$(\exists\ vs':VP\textbf{-infset} \bullet (a,vs') \in properties(x,\theta) \wedge vs' \subseteq vs)$

From axiom [consistency] we have that $vs=vs'$.

$\equiv$ **true**

$\forall\ (x,x'):(X\times X),\ a:AR,\ vs:VR\textbf{-infset},\ \theta,\theta':\Theta \bullet$
$((x,x'),(a,vs)) \in relations(\theta) \Rightarrow$
$(\exists\ vs':VR\textbf{-infset} \bullet ((x,x'),(a,vs')) \in relations(\theta) \wedge vs' \subseteq vs)$

$\theta'=add\_property((a'',vs''),x',\theta)$:

$((x,x'),(a,vs)) \in relations(\theta) \Rightarrow$
$(\exists\ vs':VR\textbf{-infset} \bullet$
$((x,x'),(a,vs')) \in relations(add\_property((a'',vs''),x',\theta)) \wedge$
$vs' \subseteq vs)$

$\equiv$ $((x,x'),(a,vs)) \in relations(\theta) \Rightarrow$
$(\exists\ vs':VR\textbf{-infset} \bullet ((x,x'),(a,vs')) \in relations(\theta) \wedge vs' \subseteq vs)$

From axiom [consistency] we have that $vs=vs'$.

$\equiv$ **true**

$\forall\ x:X \bullet decomposition(x,\theta) \subseteq decomposition(x,\theta')$

$\theta'=add\_property((a'',vs''),x',\theta)$:

$decomposition(x,\theta) \subseteq decomposition(x,add\_property((a'',vs''),x',\theta))$

$\equiv$ $decomposition(x,\theta) \subseteq decomposition(x,\theta)$

$\equiv$ **true**

QED

□

PROPOSITION 6.5 *The design move given by the generator function* ***add_relation*** *order preserves* $(\leq,\Theta)$.

Proof: We show that the generator function do not violate the axiom for the partial order.

$\forall\ \theta,\theta'{:}\Theta,\ x{:}X \bullet x \in objects(\theta) \Rightarrow x \in objects(\theta')$

θ'=add_relation((a″,vs″),(x′,x″),θ):

$x \in objects(\theta) \Rightarrow x \in objects(add\_relation((a'',vs''),(x',x''),\theta))$

$\equiv\ x \in objects(\theta) \Rightarrow x \in objects(\theta)$

$\equiv$ **true**

$\forall$ x:X, a:AP, vs:VP-**infset** •
$(x \in objects(\theta) \wedge (a,vs) \in properties(x,\theta)) \Rightarrow$
$(\exists\ vs'{:}VP\text{-}\mathbf{infset} \bullet (a,vs') \in properties(x,\theta') \wedge vs' \subseteq vs)$

θ'=add_relation(((a″,vs″),(x′,x″)),θ):

$(x \in objects(\theta) \wedge (a,vs) \in properties(x,\theta)) \Rightarrow$
$(\exists\ vs'{:}VP\text{-}\mathbf{infset} \bullet$
$(a,vs') \in properties(x,add\_relation((a'',vs''),(x',x''),\theta)) \wedge$
$vs' \subseteq vs)$

$\equiv\ (x \in objects(\theta) \wedge (a,vs) \in properties(x,\theta)) \Rightarrow$
$(\exists\ vs'{:}VP\text{-}\mathbf{infset} \bullet (a,vs') \in properties(x,\theta) \wedge vs' \subseteq vs)$

$\equiv$ **true**

$\forall\ (x,x'){:}(X\times X),\ a{:}AR,\ vs{:}VR\text{-}\mathbf{infset},\ \theta,\theta'{:}\Theta \bullet$
$((x,x'),(a,vs)) \in relations(\theta) \Rightarrow$
$(\exists\ vs'{:}VP\text{-}\mathbf{infset} \bullet ((x,x'),(a'',vs'')) \in relations(\theta') \wedge$
$vs' \subseteq vs)$

θ'=add_relation((a″,vs″),(x″,x‴),θ):

$((x,x'),(a,vs)) \in \text{relations}(\theta) \Rightarrow$
$\quad(\exists\ vs':VR\textbf{-infset} \bullet$
$\quad\quad((x,x'),(a,vs')) \in \text{relations}(\text{add\_relation}((a'',vs''),(x'',x'''),\theta)) \wedge$
$\quad\quad\quad vs' \subseteq vs)$

$\equiv\ ((x,x'),(a,vs)) \in \text{relations}(\theta) \Rightarrow$
$\quad\quad(\exists\ vs':VP\textbf{-infset} \bullet ((x,x'),(a,vs')) \in (\text{relations}(\theta) \cup$
$\quad\quad\quad\{((a'',vs''),(x'',x'''))\}) \wedge vs' \subseteq vs)$

From axiom [consistency] we have that vs=vs′

$\equiv$ **true**

$\forall\ \theta,\theta':\Theta,\ x:X \bullet \text{decomposition}(x,\theta) \subseteq \text{decomposition}(x,\theta')$

θ'=add_relation(x′,θ):

$\text{decomposition}(x,\theta) \subseteq \text{decomposition}(x,\text{add\_relation}((a'',vs''),(x'',x'''),\theta))$

$\equiv\ \text{decomposition}(x,\theta) \subseteq \text{decomposition}(x,\theta)$

$\equiv$ **true**

QED

□

PROPOSITION 6.6 *The design move given by the generator function* ***decompose*** *order preserves* $(\leq,\Theta)$*.*

Proof: We show that the generator function do not violate the axiom for the partial order.

$\forall\ \theta,\theta':\Theta,\ x:X \bullet x \in \text{objects}(\theta) \Rightarrow x \in \text{objects}(\theta')$

θ'=decompose(x′,am,θ):

$x \in objects(\theta) \Rightarrow x \in objects(decompose(x',am,\theta))$

$\equiv$ $x{:}X \bullet x \in objects(\theta) \Rightarrow x \in objects(\theta) \cup \{x''|x'':X \bullet (\exists\ a{:}AP \bullet a \in \mathbf{dom}\ am \wedge x'' \in \mathbf{dom}\ am(a))\}$

$\equiv$ **true**

$\forall\ \theta,\theta'{:}\Theta,\ x{:}X,\ a{:}AP,\ vs{:}VP\textbf{-infset} \bullet$
$(x \in objects(\theta) \wedge (a,vs) \in properties(x,\theta)) \Rightarrow$
$(\exists\ vs'{:}VP\textbf{-infset} \bullet (a,vs') \in properties(x,\theta') \wedge vs' \subseteq vs)$

θ'=decompose(x′,am,θ):

$(x \in objects(\theta) \wedge (a,vs) \in properties(x,\theta)) \Rightarrow$
$(\exists\ vs'{:}VP\textbf{-infset} \bullet (a,vs') \in properties(x,decompose(x',am,\theta)) \wedge$
$vs' \subseteq vs)$

x ∈ decomposition(x′,θ):

$\equiv$ $(x \in objects(\theta) \wedge (a,vs) \in properties(x,\theta)) \Rightarrow$
$(\exists\ vs'{:}VP\textbf{-infset} \bullet (a,vs') \in derive(am,x) \wedge vs' \subseteq vs)$

From axiom [decomp-consistency] we have that $x \notin objects(\theta)$[10]:

$\equiv$ **true**

else:

$(x \in objects(\theta) \wedge (a,vs) \in properties(x,\theta)) \Rightarrow$
$(\exists\ vs'{:}VP\textbf{-infset} \bullet (a,vs') \in properties(x,decompose(x',am,\theta)) \wedge$
$vs' \subseteq vs)$

$\equiv$ $(x \in objects(\theta) \wedge (a,vs) \in properties(x,\theta)) \Rightarrow$
$(\exists\ vs'{:}VP\textbf{-infset} \bullet (a,vs') \in properties(x,\theta) \wedge vs' \subseteq vs)$

[10]Note, that the function *derive* is not related to the partial order which is why we do not utilize that *decompose* evaluates to the result of applying *derive*.

From axiom [consistency] we have that vs=vs′.

≡ **true**

∀ (x,x′):(X×X), a:AR, vs:VR-**infset**, θ,θ':Θ •
((x,x′),(a,vs)) ∈ relations(θ) ⇒
(∃ vs′:VR-**infset** • ((x,x′),(a,vs′)) ∈ relations(θ')) ∧ vs′ ⊆ vs

θ'=decompose(x′,am,θ):

((x,x′),(a,vs)) ∈ relations(θ) ⇒
(∃ vs′:VR-**infset** • ((x,x′),(a,vs′)) ∈ relations(decompose(x′,am,θ)) ∧
vs′ ⊆ vs)

≡ ((x,x′),(a,vs)) ∈ relations(θ) ⇒
(∃ vs′:VR-**infset** • ((x,x′),(a,vs′)) ∈ relations(θ) ∧ vs′ ⊆ vs)

From axiom [consistency] we have that vs=vs′

≡ **true**

∀ θ,θ':Θ, x:X • decomposition(x,θ) ⊆ decomposition(x,θ')

θ'=decompose(x′,am,θ):

decomposition(x,θ) ⊆ decomposition(x,decompose(x′,am,θ))

x=x′:

≡ {x″|x″:X • (∃ a:AP • a ∈ **dom** am ∧ x″ ∈ **dom** am(a))} ⊆
{x″|x″:X • (∃ a:AP • a ∈ **dom** am ∧ x″ ∈ **dom** am(a))}

≡ **true**

x∈ {x″|x″:X • (∃ a:AP • a ∈ **dom** am ∧ x″ ∈ **dom** am(a))}:

$\equiv$ decomposition(x,θ) $\subseteq$ {}

$\equiv$ {} $\subseteq$ {}

else:

$\equiv$ $\forall$ x:X • decomposition(x,θ) $\subseteq$ decomposition(x,θ)

$\equiv$ **true**

QED

□

PROPOSITION 6.7 *The design move given by the generator function* ***join*** *order preserves* ($\leq$,Θ).

Proof: We show that the generator function does not violate the axiom of partial order. We do so by showing that the order is preserved between θ and θ'' for arbitrary θ' in $\theta'' = join(\theta, \theta')$. The dual case comes out of commutativity of set operations.

$\forall$ θ,θ'':Θ, x:X • x $\in$ objects(θ'') $\Rightarrow$ x $\in$ objects(θ)

θ''=join(θ,θ'):

x $\in$ objects(θ'') $\Rightarrow$ x $\in$ objects(θ)

$\equiv$ x $\in$ objects(join(θ,θ')) $\Rightarrow$ x $\in$ objects(θ)

$\equiv$ x $\in$ (objects(θ) $\cap$ objects(θ')) $\Rightarrow$ x $\in$ objects(θ)

$\equiv$ **true**

$\forall$ θ, θ'':Θ, x:X, a:AP, vs$''$:VP-**infset** •
(x $\in$ objects(θ'') $\wedge$ (a,vs$''$) $\in$ properties(x,θ'')) $\Rightarrow$
($\exists$ vs:VP-**infset** • (a,vs) $\in$ properties(x,θ) $\wedge$ vs $\subseteq$ vs$''$)

$\underline{\theta''=\text{join}(\theta,\theta')}$:

$\equiv$ (x $\in$ objects(join(θ,θ')) $\wedge$
(a,vs$''$) $\in$ properties(x,join(θ,θ'))) $\Rightarrow$
($\exists$ vs:VP-**infset** $\bullet$ (a,vs) $\in$ properties(x,θ) $\wedge$ vs $\subseteq$ vs$''$)

$\equiv$ (x $\in$ (objects(θ) $\cap$ objects(θ')) $\wedge$
(a,vs$''$) $\in$ join(properties(x,θ),properties(x,θ'))) $\Rightarrow$
($\exists$ vs:VP-**infset** $\bullet$ (a,vs) $\in$ properties(x,θ) $\wedge$ vs $\subseteq$ vs$''$)

From inspection according to the definition of join, we get:

$\equiv$ **true**

$\forall$ (x,x$'$):(X$\times$X), a:AR, vs$''$:VR-**infset**, θ,θ'':Θ $\bullet$
((x,x$'$),(a,vs$''$)) $\in$ relations(θ'') $\Rightarrow$
($\exists$ vs:VR-**infset** $\bullet$ ((x,x$'$),(a,vs)) $\in$ relations(θ) $\wedge$ vs $\subseteq$ vs$''$)

$\underline{\theta''=\text{join}(\theta,\theta')}$:

$\equiv$ ((x,x$'$),(a,vs$''$)) $\in$ relations(join(θ,θ')) $\Rightarrow$
($\exists$ vs:VR-**infset** $\bullet$ ((x,x$'$),(a,vs)) $\in$ relations(θ) $\wedge$ vs $\subseteq$ vs$''$)

$\equiv$ ((x,x$'$),(a,vs$''$)) $\in$ join(relations(θ),relations(θ')) $\Rightarrow$
($\exists$ vs:VR-**infset** $\bullet$ ((x,x$'$),(a,vs)) $\in$ relations(θ) $\wedge$ vs $\subseteq$ vs$''$)

From inspection according to the definition of join, we get:

$\equiv$ **true**

$\forall$ θ,θ'':Θ, x:X, decomposition(x,θ'') $\subseteq$ decomposition(x,θ)

$\underline{\theta''=\text{join}(\theta,\theta')}$:

decomposition(x,join(θ,θ')) $\subseteq$ decomposition(x,θ)

$\equiv$ (decomposition(x,θ) $\cap$ decomposition(x,θ')) $\subseteq$ decomposition(x,θ)

$\equiv$ **true**

QED

□

PROPOSITION 6.8 *The design move given by the generator function* ***meet*** *order preserves* ($\leq$,Θ).

Proof: We show that the generator function do not violate the axiom for the partial order. We do so by showing that the order is preserved between θ and θ'' for arbitrary θ' in $\theta'' = meet(\theta, \theta')$. The dual case comes out of commutativity of set operations.

$\forall\ \theta,\theta'':\Theta,\ \text{x:X} \bullet \text{x} \in \text{objects}(\theta) \Rightarrow \text{x} \in \text{objects}(\theta'')$

θ''=meet(θ,θ'):

x $\in$ objects(θ) $\Rightarrow$ x $\in$ objects(θ'')

$\equiv$ x $\in$ objects(θ) $\Rightarrow$ x $\in$ objects(join(θ,θ'))

$\equiv$ x $\in$ objects(θ) $\Rightarrow$ x $\in$ (objects(θ) $\cup$ objects(θ'))

$\equiv$ **true**

$\forall\ \theta,\theta'':\Theta,\ \text{x:X},\ \text{a:AP},\ \text{vs:VP-}\textbf{infset} \bullet$
(x $\in$ objects(θ) $\wedge$ (a,vs) $\in$ properties(x,θ)) $\Rightarrow$
($\exists$ vs$'$:VP-**infset** $\bullet$ (a,vs$'$) $\in$ properties(x,θ'') $\wedge$ vs$'$ $\subseteq$ vs)

θ''=meet(θ,θ'):

(x $\in$ objects(θ) $\wedge$ (a,vs) $\in$ properties(x,θ)) $\Rightarrow$
($\exists$ vs$'$:VP-**infset** $\bullet$ (a,vs$'$) $\in$ properties(x,join(θ,θ')) $\wedge$ vs$'$ $\subseteq$ vs)

$\equiv$ (x $\in$ objects(θ) $\wedge$ (a,vs) $\in$ properties(x,θ)) $\Rightarrow$
(∃ vs′:VP-**infset** •
(a,vs) $\in$ join(properties(x,θ),properties(x,θ')) $\wedge$
vs′ $\subseteq$ vs)

From inspection according to the definition of meet, we get:

$\equiv$ **true**

$\forall$ (x,x′):(X$\times$X), a:AR, vs:VR-**infset**, θ,θ'':Θ •
((x,x′),(a,vs)) $\in$ relations(θ) $\Rightarrow$
(∃ vs′:VR-**infset** • ((x,x′),(a,vs′)) $\in$ relations(θ'') $\wedge$ vs′ $\subseteq$ vs)

θ''=meet(θ,θ'):

$\equiv$ ((x,x′),(a,vs)) $\in$ relations(θ) $\Rightarrow$
(∃ vs′:VR-**infset** • ((x,x′),(a,vs)) $\in$ relations(meet(θ,θ')) $\wedge$
vs′ $\subseteq$ vs)

$\equiv$ (x,x′):(X$\times$X), a:AR, vs:VR-**infset**, θ,θ'':Θ •
((x,x′),(a,vs)) $\in$ relations(θ) $\Rightarrow$
(∃ vs′:VR-**infset** • ((x,x′),(a,vs)) $\in$ relations(θ')) $\wedge$
vs′ $\subseteq$ vs)

From inspection according to the definition of meet, we get:

$\equiv$ **true**.

$\forall$ θ,θ'':Θ, x:X • decomposition(x,θ) $\subseteq$ decomposition(x,θ'')

θ''=meet(θ,θ'):

decomposition(x,θ) $\subseteq$ decomposition(x,join(θ,θ'))

$\equiv$ $\forall$ x:X • decomposition(x,θ) $\subseteq$

(decomposition(x,θ) meet decomposition(x,θ'))

$\equiv$ **true**

QED

□

6.6 Conclusion

In this paper, we have elaborated on the notion of design lattices from Chapter 5. We have done so by taking another approach to model design representations (artefact models) and the moves which bind representation through stages of development (design moves). The approach is the *algebraic* approach from the RAISE method. In the approach, domain entities are modelled as abstract types (sorts) and functions of two different kinds are specified as function signatures ranging over the abstract types. The two kinds of functions are *observer functions* and *generator functions*. With observer functions, we observe entities of the values of abstract types. With generator functions we indicate the changes that can be made to values of the abstract types. In this paper, the generator functions represent design moves. The algebraic approach now appears from combining observer functions with generator functions in formal axioms.

As in Chapter 5, we have made a distinction between two sorts of design moves (and thus two sorts of generator functions). The former is the class of design moves *by aspect* which are modelled as functions updating artefact models; e.g. with new objects. The latter is the class of design moves *by configuration* which are modelled as binary operations on pairs of artefact models. As in Chapter 5, these binary operations are lattice join and meet.

The notion of *decomposition* has been introduced as a new sort of design move in this paper. A decomposition introduces sub–objects and associates these with a given object in a model. The function is essentially distinct from the lattice operations as the relation between object and sub–objects is represented *within* an artefact model and not as a relation between artefact models. The purpose of decomposition is to refer to parts of objects in a convenient way. However, it is not based on complex mereological issues. As an example, we do not make any restrictions to how objects are decomposed nor to their ontological status as objects falling under well–defined concepts[11]. Neither do we make any

[11] For a discussion of such issues, we refer to Chapter 8.

assumptions of how these sub–objects must relate to each other, like excluding spatial overlap.

For decompositions, it seemed plausible to suggest that a relation should hold between the properties of an object and the properties of its sub–objects. Thereby, a requirement of such a relation could be included in the axiom for partial order. We have explored various criteria for such a relation and we found that it is not plausible to define such a relation. The reason is that some properties — like the length of a beam — may not maintain their values in sub–objects. Other properties — like the colour of a house — may be considered dominating for some sub–objects compared to others, but a consistent theory of essential parts is here needed for clarification. We have argued that a consistent criteria, not obstructing the process of the design practitioner, cannot be defined; this based on the present ontological status on which this paper has been founded.

The notions of artefact models and design moves are intended to span a lattice structure as were the notions in Chapter 5. In Chapter 5, we showed that this was indeed the case. However, in this paper, the specification is different, although the intrinsic structure is the same as in Chapter 5. We have shown that the generator functions do not violate the axiom for partial order. Combining the proofs and considerations from Chapter 5, we have that artefact models and moves satisfy lattice criteria.

Future work can turn in various directions. One direction may try formally to relate the algebraic specification to mathematically well–founded algebras or calculi like *Lambda–calculus*, Cardelli's $F_{<:}$, etc. An approach could be to encode the algebraic specification in these calculi. Thereby, the mathematical properties of the specification could be explored. Also, the notion of artefact models could be related to category theory by defining the notion of artefact models as a category. Thereby, further steps towards a mathematically founded design algebra will be taken.

Another direction for future work could be to specify convenient languages for design representation. A basic core–language is presented in Chapter 7. However, other sorts of languages may be explored. An important step in such future work is to apply such languages as means for expression in cases of real designing.

Chapter 7

Semantic parameterized interpretation as a foundation for conceptual design systems

Abstract: This paper suggests a software architecture for conceptual design systems. We claim that today's design systems lack of dynamics in two respects. The former is that objects usually cannot be specialised incrementally. The latter is that introducing new sorts of properties and relations usually requires a large amount of programming, code restructuring, and recompilation.

The suggested software architecture aims at introducing the desired dynamics by means of what we shall call semantic parameterized interpretation. Conceptual design models are expressed in a modelling language $\mathcal{L}_M$ and interpreted according to a semantics written in a specification language $\mathcal{L}_S$. The semantics specifies how names of properties and relations are to be interpreted. The result of the interpretation is what we call a *view* of the model. A view is expressed in one of many possible target languages $\mathcal{L}_{T_i}$. A view can be a representation of visualisation, a database scheme for bill–of–material calculations, equations for stress analysis, etc. Each semantics defines a specific sort of view.

A conceptual model contains: Object designators which are bound to sets of properties, relations between objects, and decompositions of objects into sub–objects.

New names for properties and relations can be introduced freely by expressing their meaning in the semantic language. This principle stands in contrast to the object–oriented principle which founds most of today's design systems relying on pre–defined class hierarchies for design objects.

We specify the syntax and denotational semantics of the languages involved. Furthermore, we present a formal specification of the software architecture which is founded on the principle of semantic parameterized interpretation.

However, the presented languages are not advanced in any sense, but simply examples. The contribution, we believe, is the principle of semantic parameterized interpretation as a new software architecture for conceptual design systems.

Notation

The following is a list of some symbols which are repeatingly used in the paper.

$\mathcal{L}_M$	Artefact specification language
m	Range over artefact specifications written in $\mathcal{L}_M$
$\mathcal{L}_S$	Family of semantic specification languages However, $\mathcal{L}_S$ is also used to denote a semantic specification language in general
$\mathcal{L}_{S_i}$	Family member of $\mathcal{L}_S$
s	Range over semantics specifications
$\mathcal{L}_T$	Family of target languages
$\mathcal{L}_{T_i}$	Instances of the family of target languages.
t	Range over semantics specifications

7.1 Introduction

The introduction of object–orientation has had a tremendous impact on the foundation and software architectures for computer aided design systems (CAD systems). Such systems are today mostly object–oriented and founded on built–in class hierarchies for the kind of objects which can be added to a design. Examples are classes of walls, ceiling, beams, doors, and furniture objects. A design — in this respect — is a configuration of objects, and designing is a process of adding objects of classes to a configuration.

Class–based languages makes the majority of object–oriented languages. In class–based languages, information is represented in objects by means of properties. Objects are instances of classes which contain methods for representing and manipulating object data, [1].

Some methods are intended to capture kinds of semantic aspects of the objects. An example is a method for visualising an object on screen. The method expresses the visual properties of objects of that class by means of graphical and geometrical values like lines, circles, and colour values.

We claim that object–oriented design systems lack of dynamics in two respects. The first is that the systems do not support incremental design; i.e. a process in which objects of the current design model are being specialised incrementally according to their kinds. In object–orientation, each object is "born" with a number of properties; namely those specified by its class. The second respect is that introducing new sorts of properties or relations means re–organising the class hierarchy which may require a large amount of programming, code re–structuring, and re–compilation of the design program code. The latter may have the impact that over time, the class system gets messy.

In [64, 67, 75, 167, 62] and in Chapter 5 and Chapter 6, foundations for conceptual design systems have been explored. It is argued that properties of objects and relations between objects are fundamental ontological entities for artefact description. By an artefact, we understand a man–made physical object. Most mechanical and architectural designing concerns artefacts. However, properties and relations do not necessarily have to be "selected" from a pre–defined class hierarchy.

In another sense, this paper adds another dimension to the discussion of interoperability in construction and design of buildings.

7.1.1 Design models and views: a few examples

Consider a design conception of a beam object. The fundamental knowledge of such a beam is the ontological intrinsic properties like dimensions and material. Such knowledge can to some extend be expressed in a formal specification. For our beam object (let us call it *mybeam*), we could give the specification:

```
model
  mybeam :  (sort={beam},
             length={2400},
             height={600},
             width={800},
             material={wood})
```

We call such a specification an *artefact model*, as the objects with which we are concerned, are artefacts. The language in which it is written, we call a *modelling language*, denoted $\mathcal{L}_M$.

An artefact model specifies the properties of uniquely identified objects; here the singleton set of the object named *mybeam*. Properties are represented by expressions like *length*=$\{$*2400*$\}$, where the name *length* is called the attribute and *2400* is called the property value (or just the *value*).

Note, that we have included a property with the attribute *sort*. The reason is that it is much easier to distinguish the objects on their sort (an overall concept) than making a long list of properties aiming a defining such distinctions. A similar approach is used in object–oriented languages where we have the notion of a *class name* in addition to the type structure inside the class definition. However, the attribute *sort* is optional and considered to by at the same level as all other attributes. We might even call it something else or omit it if it is not needed.

From an artefact model (m:$\mathcal{L}_M$), we can observe the set of object identifiers (xs:X-**set**) in the model. Also, from an artefact model and an object identifier, we can observe the set of properties (ps:A $\xrightarrow[m]{}$V) ascribed to that object in the model. The set of properties of an object is modelled as a mapping from attributes (a:A) to property values (v:V).

type
  $\mathcal{L}_M$, X, A, V

value
  objects: $\mathcal{L}_M \rightarrow$ X-**set**,

properties: $\mathcal{L}_M \times \mathrm{X} \rightarrow (\mathrm{A} \xrightarrow[m]{} \mathrm{V})$

Let m be the artefact model which includes the object *mybeam*, then *objects*(m) gives {*mybeam*} and *properties*(m,*mybeam*) gives:

[sort $\mapsto$ beam, length $\mapsto$ 2400, height $\mapsto$ 600, width $\mapsto$ 800, material $\mapsto$ wood]

The two observer functions are modified later in Section 7.2.1).

The specification only states the ontological knowledge intimately (intrinsically) related to the object, and thus not behaviour, function, potential use, or how the object looks like from various angles. Such knowledge are extrinsic to the object and depends on the perspective we put on the artefact model specifying it. As shown on Figure 7.1, various different perspectives exists. We can visualize the beam in front view or in isometric view, we can illustrate the load–theoretic issues which hold for the beam, present MatLab programs for calculating internal stress, or we can even present another artefact model being the result of scaling the beam's proportion by a factor $\frac{1}{2}$.

The different perspectives are the results — indirectly or directly — of interpreting the artefact model in different ways. In fact an artefact model can be interpreted in as many ways as there are pieces of information which can be deduced from it.

An interpretation of an artefact model is performed by translating objects in the artefact model to the meanings of terms denoting properties of these objects. However, even though the names *sort*, *length*, *height*, etc., might make sense in our normal understanding of artefacts, they do not in computer programs until we specify the meaning of them.

A specification which defines the meaning of property names, we call a *semantics*, and the language in which it is written we call a *semantic language* denoted $\mathcal{L}_S$.

The semantics for getting a front view interpretation of objects like *mybeam* could be given by the specification[1]:

[1] Note the use of *pre–fix* notation.

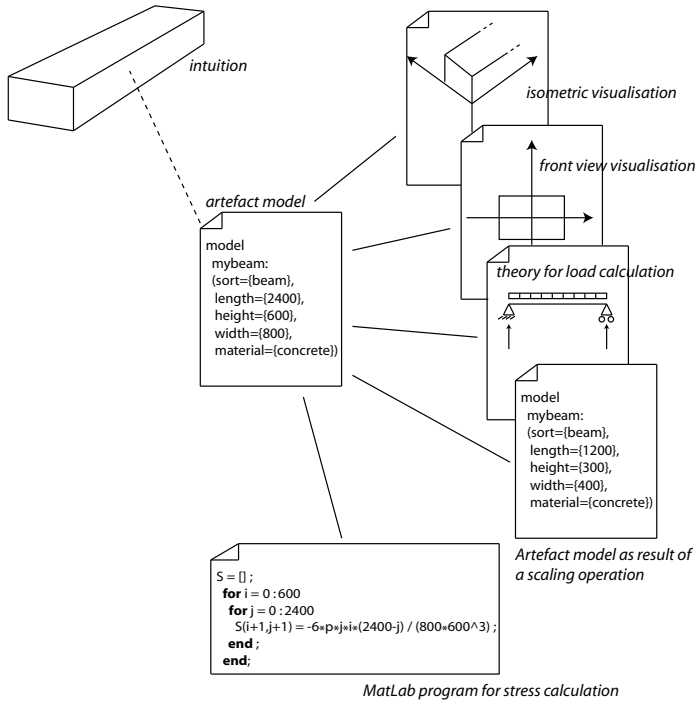


Figure 7.1: Multiple views on a model.

```
template
 (sort={beam}) :
  {
   (command "rectangle" (list (/ -width 2)) (/ -height 2)))
                        (list (/ width 2)   (/ height 2)))
  }
```

Expressions like ($sort$=$\{beam\}$) are lists (here a singleton) of properties similar to the list given for the object *mybeam* in the artefact model. In a semantics, we call such an expression a *property pattern*. The semantics states that the meaning of an object with attribute *sort* and value *beam* is the rectangle command expression enclosed in the curly parenthesis.

From a semantics, we can observe the set of property patterns, and from a semantics and a property pattern, we can observe the meaning which is expressed in $\mathcal{L}_T$.

type

$\mathcal{L}_S$, $\mathcal{L}'_T$, A, V

value
patterns: $\mathcal{L}_S \rightarrow (\text{A} \xrightarrow[m]{} \text{V})$,
target: $\mathcal{L}_S \times (\text{A} \xrightarrow[m]{} \text{V}) \rightarrow \mathcal{L}'_T$

We recognize that the pattern ($sort = \{beam\}$) complies with the properties of *mybeam* in the artefact model presented. It does so in the sense that the set of properties in the pattern (here a singleton set) is a subset of the set of properties of the object *mybeam*. We assume that the "best" target is selected in case of ambiguity of any kind. A more sophisticated and fine grained approach is given in Section 7.5 and Section 7.9.

Now, assume that we name the semantic specification s and the artefact model m. A first, partial interpretation of *mybeam* (and thus of the artefact model) is given by:

target(s,properties(m,mybeam))

and the result is:

```
(command "rectangle" (list (/ -width 2)) (/ -height 2)))
                     (list (/ width 2)   (/ height 2)))
```

We call such a representation a *view* (or a *target*). The language in which it is written is called a *target language* denoted $\mathcal{L}_{T_i}$.

This view is, however, *unsaturated* in the sense that it contains the placeholders *width* and *height*, as well as algebraic expressions to be evaluated[2].

Substituting free variables and evaluating arithmetic sub–expressions yields the resulting representation of a front–view perspective on the artefact model. The property set of an object is here considered a substitution of attributes with property values. For *mybeam*, we have the substitution:

[sort ↦ beam, length ↦ 2400, height ↦ 600, width ↦ 800, material ↦ wood]

[2]Frege used the term "*unsaturated*" about functions which become saturated by means of applying their arguments. In this paper, the term is used in a different sense, though inspired by Frege. We shall discuss the distinction between saturated and unsaturated views further in Section 7.4.

which yields:

```
(command "rectangle" (list -400 -300) (list 400 300))
```

This resulting view is a valid expression in the graphical language *AutoLISP* which is a built–in language of the design system AutoCAD. Entering the view specification into AutoCAD makes this tool display a visualisation of the object as shown on Figure 7.2[3]. We should note that AutoLISP uses a *pre–fix* notation for arithmetic expressions which is why we used this notation earlier.

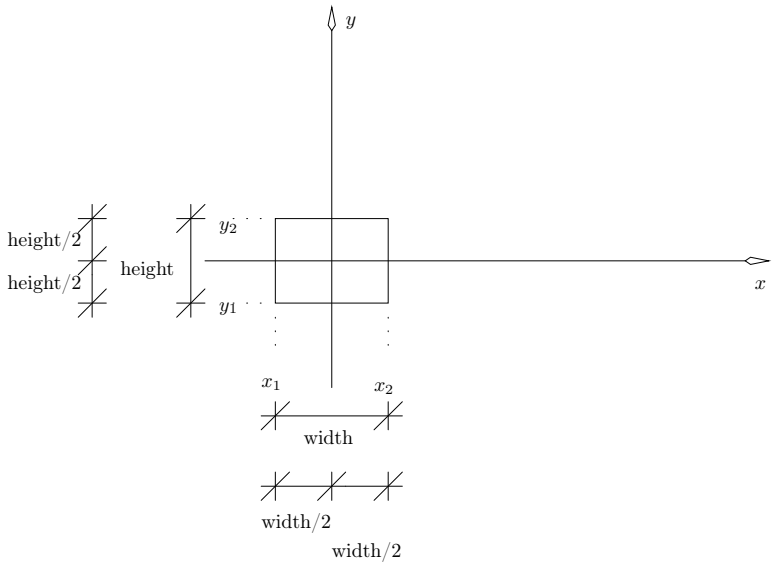


Figure 7.2: Front view visualization of `mybeam`.

A semantics which yields an *oblique projection* view of the artefact model could be:

[3]Measurements are only shown for convenience.

```
template
 (sort={beam}) :
  {
   (command "rectangle" (list (/ -width 2) (/ -height 2))
                        (list (/ width 2) (/ height 2)))
   (command "line" (list (/ -width 2) (/ height 2))
                   (list (+(/ -width 2) length) (+(/ height 2) height)) "")
   (command "line" (list (+(/ -width 2) length) (+(/ height 2) height))
                   (list (+(/ width 2) length) (+(/ height 2) height)) "" )
   (command "line" (list (/ width 2) (/ height 2))
                   (list (+(/ width 2) length) (+(/ height 2) height)) "" )
   (command "line" (list (/ width 2) (/ -height 2))
                   (list (+(/ width 2) length) (/ height 2)) "" )
   (command "line" (list (+(/ width 2) length) (/ height 2))
                   (list (+(/ width 2) length) (+(/ height 2) height)) "" )
  }
```

The interpretation of the model according to this semantics gives the view[4]:

```
(command "rectangle" (list -400 -300) (list 400 300))
(command "line" (list -400 300) (list (+(-400 2400)) (+ 300 600)) "")
(command "line" (list (+ 400 2400) (+ 300 600)) (list (+ 400 2400)
                                                      (+ 300 600)) "")
(command "line" (list 400 300) (list (+ 400 2400) (+ 300 600)) "")
(command "line" (list 400 -300) (list (+ 400 2400) 300) "")
(command "line" (list (+ 400 2400) 300) (list (+ 400 2400) (+ 300 600)) "")
```

The isometric and the front view relate in the sense that the latter is written in a sub–language of which the former is written. Also, they relate in the sense that the former is simply a sub–expression of the latter.

We can also define views which are not intended for graphical display of the form of the artefact. Consider for example[5]:

```
template
 (sort={beam}) :
  {
   0.5*width*height*length kg
  }
```

Applying this semantics on the artefact model gives the view:

```
3200 kg
```

I.e. we get a sequence of characters which express the mass of such beams.

[4]For clarity, we have avoided to completely evaluate the expressions into values

[5]Laminated wood usually have a mass of $0.5 * Akg/m$, where A is the cross area of such beams [111].

A more advanced example could be a semantics which defines a view representing a MatLab programs for calculating internal stress of such beams. Consider the following semantics:

```
template
 (sort={beam}) :
  {
   S=[];
   for i = 0:height
     for j = 0:length
       S(i+1,j+1)=-6*p*j*i*(length-j)/(width*height^3);
     end;
   end;
  }
```

The semantics states that the meaning of an object which complies with the given property pattern, is a MatLab programs which gives a matrix of stress values[6]. The view is: Applying the semantics on the artefact model gives the view:

```
S=[];
for i = 0:600
  for j = 0:2400
    S(i+1,j+1)=-6*p*j*i*(2400-j)/(800*600^3);
  end;
end;
```

7.1.2 Semantic parameterized interpretation

The class of design systems we outline in this paper rely on a principle we call *semantic parameterized interpretation*. The design systems belonging to the class follow a common structure — they have the same *software architecture*. By a software architecture we understand the set of components and the interfaces between components which together outlines a system structure [29].

Definition 7.1 (Semantic parameterized interpretation) By semantic parameterized interpretation, we understand interpretation of artefact models according to a semantics.

[6]The values are calculated on basis of Navier's formula $\delta = -\frac{M}{I}y$, where M is the force equation $M = \frac{1}{2}pLx - \frac{1}{2}pLx^2$ and I is the *moment of inertia* for beams with a rectangular cross section: $\frac{1}{12}bh^3$. x and y are coordinate factors, L is the beam length, b and h are the width and height of the beams, respectively, and p is the density of the material.

A semantics binds sets of properties to target language expressions. Interpretation is now made in two stages. The first stages of the interpretation is gives the target expressions for each object in the model. Each target expression in the set is a view of the individual objects. However, the target expressions may be *unsaturated* in the sense that they contain free terms and unevaluated expressions. The second stage of the interpretation aims at: (i) substituting such free terms with property values to which the terms as attributes in the artefact model, are bound, (ii) evaluating expressions, and (iii) combining the resulting views into one.

The second part is in fact a kind of *term re–writing* and evaluation [54]. It requires what we shall call a *calculus semantics* $\mathcal{CS}$ which basically is a canonical set of rewriting rules which include substitution. We call it a *calculus semantics* as it specifies how to interpret the basic arithmetic expressions belonging to the general calculus underlying the given target language.

Semantic parameterized interpretation is thus based on:

$\mathcal{L}_M$: a language for expressing artefact models.

$\mathcal{L}_S$: a language for specifying the semantics of artefact models.

$\sum_i \mathcal{L}_{T_i}$: a family of target languages for expressing views.

$\mathcal{CS}$: a calculus semantics.

Parsing an artefact model (m:$\mathcal{L}_M$) gives an internal representation (θ:Θ). In one sense the type Θ is just another abstract syntax of $\mathcal{L}_M$. In another sense, we can see it as a sort of environment and thus interpreting an artefact model means building up such an environment. In Chapter 5 and designalgebra, we called (θ:Θ) an artefact model. In this paper, we call it an *object environment* in order to avoid confusion with artefact models expressed following the syntax given abstractly in Section 7.2.1 and concretely in Section 7.11. More precisely:

Definition 7.2 (Object environment) By an object environment, we understand a record of two mappings: One which maps object identifiers to sets of properties ascribed to objects, and one which maps pairs of object identifiers to sets of relations ascribed to object pairs.

□

Parsing a semantics (s:$\mathcal{L}_S$), gives a semantic environment (ρ:ENV).

Definition 7.3 (Semantic environment) By a semantic environment, we understand a mapping from sets of properties (property patterns) to unsaturated expressions in a target language.

□

The production of a view (t:$\mathcal{L}_{T_i}$), as the result of full interpretation of an artefact model (m:$\mathcal{L}_M$), according to a semantics (s:$\mathcal{L}_S$) and a calculus semantics ($\mathcal{CS}$:CS), is performed in the following steps:

$\mathcal{M}$: translates an artefact model (m:$\mathcal{L}_M$) into an object environment (θ:Θ).

$\mathcal{SM}$: translates a semantics (s:$\mathcal{L}_S$) into a semantic environment (ρ:ENV).

$\mathcal{I}'$: interprets each object according to the semantics (s:$\mathcal{L}_S$). The result is a set of target expressions (tz:$\mathcal{L}'_{T_i}$).

$\mathcal{CS}$: saturates and combines such target expressions[7].

Figure 7.3 depicts the principle of semantic parameterized interpretation.

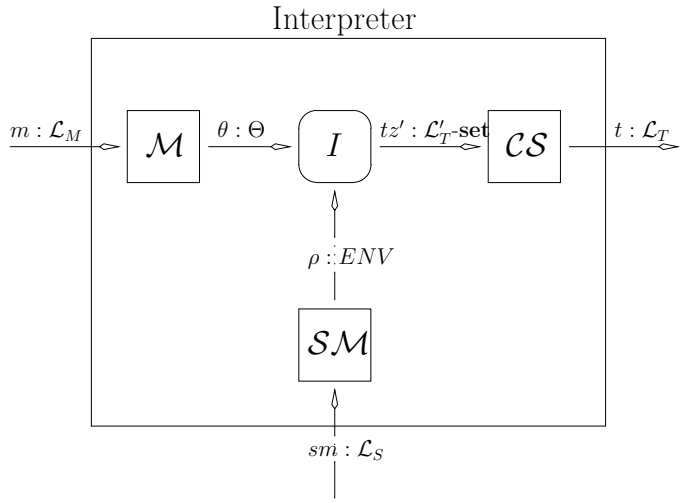


Figure 7.3: Semantic parameterized interpretation.

[7]We shall later see that in fact the function for combining targets is a parameter to the calculus semantics.

Since a semantics written in $\mathcal{L}_S$ contains target language expressions, we may need to refer to a semantics which complies with a certain target language. We do so by means of indices or names. In general, $\mathcal{L}_{S_i}$ is the semantic language for views in the target language $\mathcal{L}_{T_i}$. We let the generic $\mathcal{L}_S$ denote the semantic language for views expressed in arbitrary target languages.

7.2 The modelling language $\mathcal{L}_M$

We follow the principles of incremental design, presented in Chapter 4, Chapter 5, Chapter 6, and in the BAS•CAAD project [64, 67, 75, 167], by emphasizing properties. For ontological completion we include relations as well. However, relations play a minor rôle in this paper as they make the general interpretation much more complex. Section 7.9 discuss this in more detail and suggests some accommodations for including relations.

A property is represented by an attribute (a:A) and a value set (vs:VS). In order to distinguish identical sets of properties ascribed to distinct objects in a design, we make use of object identifiers (x:X). Of second importance come relations, which are represented in a similar manner.

We call the approach *property–orientation* in contrast to *object–orientation*. An essential difference between property–orientation and the class–centered approach of object–orientation is that property–orientation maintains the property of subsumption (see Section 7.5). In [1], this is formulated as a distinction between *sub–classing* and *sub–typing*. Sub–typing relates two types based on the structure of the type definitions. Sub–classing relates two types based on the names of the types. Thus, in sub–classing, we can have two classes which are identical in the sense that their objects have the same properties, though still not related by subsumption.

Also, sub–typing in property–orientation possess the property of *anti–symmetry* unless we consider coercion which translates values between distinct types. We shall follow the principle of sub–typing (without coercion), as we consider two objects with identical sets of properties to have the same meaning, though not to be identical.

We define the language $\mathcal{L}_M$ for expressing designs as artefact models. All model specifications begin with the keyword model. After that comes a sequence of object declarations and relation declarations. An object declaration ascribes a collection of properties to an object. Furthermore, a list of sub–objects may be ascribed the object as well. These sub–objects represent a decomposition of the

object in question (see Chapter 8). A relation declaration ascribes a collection of relations to a pair of objects.

There are two ways of expressing value sets: (i) as a comma separated sequence of values, and (ii) as an interval. Values (v:**V**) are either numerals (n:**Num**), property value names (vp:**VP**), infimum (inf) or negative infimum (neginf).

Models can be combined with two operations meet $\sqcup$ and join $\sqcap$. The intuition behind the two operations is as follows. The join of two models gives a model which has the intersection set of objects, the intersection set of sub–objects, the intersection set of attributes for each common object, and the union set of values for common attributes of common objects. The meet of two models gives a model which has the union set of objects, the union set of sub–objects, the union set of attributes for each common object, and the intersection set of values for common attributes of common objects. We call the two operations *lattice operations* as they define a lattice structure in which nodes are models and specialisation/generalisation defines the ordering. The principle is called *Design Lattices* and has been introduced in Chapter 5 and Chapter 6 using two different approaches. The present paper mostly follows the approach described in Chapter 6, although simplifications concerning representation of value sets are applied.

The two operations add a second level of sentence separations in the modelling language, where the first level is bound to ";". The operations *meet* and *join* have higher precedence than ";".

7.2.1 Syntax

We define the following syntactical categories:

m	$\in$	$\mathcal{L}_M$	artefact models
s	$\in$	$\mathbf{S}$	object/relation declarations
p	$\in$	$\mathbf{P}$	property expressions
d	$\in$	$\mathbf{X}^*$	decomposition expressions
vs	$\in$	$\mathbf{VS}$	property value sequences
v	$\in$	$\mathbf{V} = \mathbf{VP} \cup \mathbf{Num} \cup \{\mathsf{inf},\mathsf{neginf}\}$	property values

Num is the syntactic category for numerals and **VP** is the syntactic category

for property values not being numerals. The language $\mathcal{L}_M$ is fully declarative so order is only of importance in case of identifier doublets. Where sufficient, instances of the syntactical domains may be indexed in order to avoid ambiguity. Furthermore, (x:X) range over object identifiers, and (a:A) over attributes. The abstract syntax of $\mathcal{L}_M$ is as follows:

$$
\begin{array}{rcl}
m & ::= & \mathsf{model}\ s \\
 & | & m_1 \sqcap m_2 \\
 & | & m_1 \sqcup m_2 \\
s & ::= & x : (\ p\ ) \\
 & | & x : (\ p\ )\{\ d\ \} \\
 & | & (x_1, x_2) : (\ p\ ) \\
 & | & s_1;\ s_2 \\
p & ::= & a = \{vs\} \\
 & | & a = [v_1, v_2] \\
 & | & p_1,\ p_2 \\
d & ::= & x \\
 & | & d_1,\ d_2 \\
vs & ::= & v \\
 & | & v_1,\ v_2 \\
v & ::= & n \\
 & | & \mathsf{inf} \\
 & | & \mathsf{neginf} \\
 & | & v_p
\end{array}
$$

Parsing an artefact model (m:$\mathcal{L}_M$) builds up an object environment (θ:Θ). The object environment contains two mappings: *objects* and *relations*. The first maps object identifiers to pairs ((σ,xs):$\Sigma\times$X-**set**) of which (σ:Σ) is a mapping from attributes to value sets (vs:VS), and (xs:X-**set**) is a set of sub–objects. The second maps pairs of object identifiers to a map from attributes to values sets in a similar manner. Contrary to in Chapter 6, we shall not make any type distinction between values meant for expressing properties and relations. We have:

```
type
  Θ ::
    objects: X ~m~> (Σ × X-set)
    relations: (X×X) ~m~> Σ,
  X,
  Σ = A ~m~> VS,
  A,
  VS == Intv(lo:V,hi:V) | Seq(sq:V-set),
  V == Number(n:Num) | Name(pn:VP) | Inf | NegInf,
  VP,

value
  ⊥ : VS = Seq({})
```

The value ⊥ represents *absurdum*; i.e. the impossible/conflicting value of a property. It may appear in the result of applying lattice meet on artefact models which are in conflict concerning a property of a common object.

The structure gained by *objects* of Θ can be considered a tree or a forest[8] structure where the nodes are object identifiers (x:X) of the model and nodes may be annotated with property information (σ:Σ). The structure gained by *relations* relates pairs of such nodes in the tree orthogonally on the tree ordering. An example is:

```
value
  x1, x2, x3, x4, x5, x6, x7, x8, x9,
  σ1, σ2, σ3, σ4, σ5, σ6, σ7, σ8, σ9,
  σr1, σr2, σr3,

  θ : Θ = mk_Θ([ x1 ↦ (σ1,{x2,x3,x4}),
                 x2 ↦ (σ2,{}),
                 x3 ↦ (σ3,{x5,x6}),
                 x4 ↦ (σ4,{}),
                 x5 ↦ (σ5,{}),
                 x6 ↦ (σ6,{}),
                 x7 ↦ (σ7,{x8,x9}),
                 x8 ↦ (σ8,{}),
                 x9 ↦ (σ9,{}) ])
```

The decomposition hierarchy is depicted in Figure 7.4.

[8] if no unique object binds upwards.

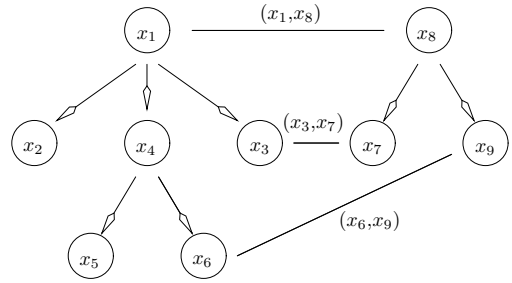


Figure 7.4: Decomposition hierarchy.

We will need to extract the first and second component in the pair ($\Sigma\times$ X-**set**) separately many times:

value
  fst: $\Sigma \times$ X-**set** $\rightarrow \Sigma$
  fst(σ,xz) $\equiv \sigma$,

  snd: $\Sigma \times$ X-**set** $\rightarrow$ X-**set**
  snd(σ,xz) $\equiv$ xz

7.2.2 Semantics

Object environments are built through a parsing process which applies a set of semantic functions. These have the signatures:

value
  $\mathcal{M}$: $\mathcal{L}_M \xrightarrow{\sim} \Theta$,
  $\mathcal{S}$: **S** $\rightarrow (\Theta \rightarrow \Theta)$,
  $\mathcal{P}$: **P** $\rightarrow \Sigma \rightarrow \Sigma$,
  $\mathcal{D}$: **D** $\rightarrow$ X-**set**,
  $\mathcal{VS}$: **VS** $\rightarrow$ V-**set**,
  $\mathcal{V}$: **V** $\rightarrow$ V

and the definitions:

$$
\begin{array}{rcl}
\mathcal{M}[\![\text{model } s]\!] & = & \mathcal{S}[\![s]\!] [\,] \\
\mathcal{M}[\![m_1 \sqcap m_2]\!] & = & \text{join}(\mathcal{M}[\![m_1]\!], \mathcal{M}[\![m_2]\!]) \\
\mathcal{M}[\![m_1 \sqcup m_2]\!] & = & \text{meet}(\mathcal{M}[\![m_1]\!], \mathcal{M}[\![m_2]\!]) \\
\mathcal{S}[\![x : (p)]\!] \theta & = & \text{mk\_}\Theta((\text{objects } \theta) \dagger [x \mapsto (\mathcal{P}[\![p]\!] [\,], \{\})], \\
 & & \qquad \text{relations } \theta) \\
\mathcal{S}[\![x : (p)\{ d \}]\!] \theta & = & \text{mk\_}\Theta((\text{objects } \theta) \dagger [x \mapsto (\mathcal{P}[\![p]\!] [\,], \mathcal{D}[\![d]\!])], \\
 & & \qquad \text{relations } \theta) \\
\mathcal{S}[\![(x_1, x_2) : p]\!] \theta & = & \text{mk\_}\Theta(\text{objects } \theta, \\
 & & \qquad (\text{relations } \theta) \dagger [(x_1, x_2) \mapsto (\mathcal{P}[\![p]\!] [\,])]) \\
\mathcal{S}[\![s_1 \,;\, s_2]\!] & = & \mathcal{S}[\![s_2]\!] \odot \mathcal{S}[\![s_1]\!] \\
\mathcal{P}[\![a = \{ vs \}]\!] \sigma & = & \sigma \dagger [a \mapsto \text{Seq}(\mathcal{VS}[\![vs]\!])] \\
\mathcal{P}[\![a = [v_1; v_2]]\!] \sigma & = & \sigma \dagger [a \mapsto \text{Intv}(\mathcal{V}[\![v_1]\!], \mathcal{V}[\![v_2]\!])] \\
\mathcal{P}[\![p_1 \,,\, p_2]\!] & = & \mathcal{P}[\![p_2]\!] \odot \mathcal{P}[\![p_1]\!] \\
\mathcal{D}[\![x]\!] & = & \{x\} \\
\mathcal{D}[\![d_1 \,,\, d_2]\!] & = & \mathcal{D}[\![d_1]\!] \cup \mathcal{D}[\![d_2]\!] \\
\mathcal{VS}[\![v]\!] & = & \mathcal{V}[\![v]\!] \\
\mathcal{VS}[\![vs_1, vs_2]\!] & = & \mathcal{VS}[\![vs_1]\!] \cup \mathcal{VS}[\![vs_2]\!] \\
\mathcal{V}[\![n]\!] & = & \text{Number}(n) \\
\mathcal{V}[\![\text{inf}]\!] & = & \text{Inf} \\
\mathcal{V}[\![\text{neginf}]\!] & = & \text{NegInf} \\
\mathcal{V}[\![v_p]\!] & = & \text{Name}(v_p)
\end{array}
$$

where *join* and *meet* are defined in Appendix 7.13, and $\odot$ is functional composition (overloaded for Θ and Σ)[9]:

value
  $\odot$: $(\Theta \to \Theta) \times (\Theta \to \Theta) \to (\Theta \to \Theta)$
  f $\odot$ g $\equiv$ λ θ:Θ • g(f(θ)),

  $\odot$: $(\Sigma \to \Sigma) \times (\Sigma \to \Sigma) \to (\Sigma \to \Sigma)$,
  f $\odot$ g $\equiv$ λ σ:Σ • g(f(σ)),

[9] We use the symbol $\odot$ in order not to confuse with the map composition operation $\circ$ in RSL.

In stead of applying functional composition by $\odot$ we could have settled with simple map overriding. For $\mathcal{S}$, the meaning would then be $\mathcal{S}[\![\ s_1\ ]\!] \dagger \mathcal{S}[\![\ s_2\ ]\!]$. This would give the same result as the semantics of s_2 does not depend on the semantics of s_1. Functional composition is chosen for simplicity, in case of changes to the language, and in order to be faithful to common denotational practice.

If *join* or *meet* are applied on a pair of models, represented as object environments, we require that each of these environments are wellformed. This is the only reason why the semantic function $\mathcal{M}$ is partial.

However, $\mathcal{M}$ does not curry (θ:Θ) (i.e. it does not have signature $\mathcal{L}_M \to (\Theta \to \Theta)$). The reason is that the two lattice operations are associative operations which combine pairs of models; they do not build up object stores alternately by means of overriding and updating. Therefore, $\mathcal{M}$ calls $\mathcal{S}$ with the empty object environment at top level.

7.2.3 Wellformedness of models

All artefact models (m:$\mathcal{L}_M$) can be represented by an object environment (θ:Θ). The language $\mathcal{L}_M$ is pure declarative and does not contain arithmetic or boolean expressions to be evaluated. However, not all object environments are wellformed in the sense that they can be interpreted into proper views. Thus, wellformedness of artefact models is turned into wellformedness of object environments. It is defined hierarchically over the structure of Θ. Basically, we require that values in a sequence are either all numbers (n:Num) or all names (s:VP), that intervals are wellformed, that relations are non–reflexive, and that decompositions are non–cyclic.

```
value
  wf_Θ: Θ → Bool
  wf_Θ(θ) ≡
    (∀ x:X • x ∈ dom objects(θ) ⇒
       wf_Σ(fst(objects(θ)(x)))) ∧
    (∀ x,x′:X • (x,x′) ∈ dom relations(θ) ⇒
       (wf_Σ(relations(θ)(x,x′)) ∧ x≠x′)) ∧
    nocycles([ x ↦ xz | x:X, xz:X-set •
       x ∈ dom objects(θ) ∧
       xz=snd(objects(θ)(x)) ])

  wf_Σ: Σ → Bool
```

```
wf_Σ(σ) ≡
   (∀ a:A • a ∈ dom σ ⇒ wf_VS(σ(a))),

wf_VS: VS → Bool
wf_VS(vs) ≡
   case vs of
      Intv(Number(n),Number(n′)) → n≤n′,
      Intv(Number(n),Inf) → true,
      Intv(NegInf,Number(n′)) → true,
      Intv(NegInf,Inf) → true,
      Seq(vs′) →
       (∀ v, v′:V • {v,v′} ⊆ vs′ ⇒
            (∃ n,n′:Num • v=Number(n) ∧ v′=Number(n′)) ∨
            (∃ s, s′:VP • v=Name(s) ∧  v′=Name(s′)))
      _ → false
   end,

nocycles: X ~m~> X-set → Bool
nocycles(xm) ≡
   (∀ x:X • x ∈ dom xm ⇒
      x ∉ {x_sub|x_sub:X • is_super(x,x_sub)(xm)}),

is_super: X × X → (X ~m~> X-set) → Bool
is_super(x,x′)(xm) ≡
   x′ ∈ xm(x) ∨
      (∃ x″:X • x″ ∈ xm(x) ∧ is_super(x″,x′)(xm))
```

For completeness, we have stated wellformedness criteria concerning relations, well aware that these are not treated in this paper; besides in Section 7.10.1.

7.3 The semantic language $\mathcal{L}_S$

We introduce the language $\mathcal{L}_S$ for specifying the meaning of artefact models.

A semantic specification is a sequence of pairs, of which the first component is a property pattern and the second component is an unsaturated target expression. Such a pair we call a *template*.

By a property pattern, we understand a specification of a set of properties as in artefact models. However, in $\mathcal{L}_S$ such a specification is considered a pattern as

we are going to match these with the properties ascribed to objects in artefact models. We use the same syntax for property patterns in $\mathcal{L}_S$ as for property sets in $\mathcal{L}_M$. The associated unsaturated target expression is enclosed in curled parentheses and represents the partial meaning of objects matching the property pattern.

7.3.1 Syntax

In addition to the syntactical categories of $\mathcal{L}_M$, we introduce the following:

sm	$\in$	$\mathcal{L}_S$	semantic specifications
tm	$\in$	**TM**	templates

Thus, we reuse the categories **P**, **A**, **VS** and **V** of $\mathcal{L}_M$. In addition, t range over target language expressions (possibly unsaturated).

The abstract syntax of $\mathcal{L}_S$ is as follows:

$$
\begin{array}{rcl}
sm & ::= & \mathsf{template}\ tm \\
 & | & sm_1;\ sm_2 \\
tm & ::= & (\ p\ )\ :\ \{\ t\ \} \\
 & | & tm_1;\ tm_2 \\
p & ::= & a = \{vs\} \\
 & | & a = [v_1, v_2] \\
 & | & p_1,\ p_2 \\
d & ::= & x \\
 & | & d_1,\ d_2 \\
vs & ::= & v \\
 & | & vs_1,\ vs_2 \\
v & ::= & n \\
 & | & \mathsf{inf} \\
 & | & \mathsf{neginf} \\
 & | & v_p
\end{array}
$$

As for $\mathcal{L}_M$, we allow instances of the syntactical domains to be indexed where sufficient. Parsing a semantics (sm:$\mathcal{L}_S$) builds up a semantic environment (ρ:ENV) which maps property patterns (σ:Σ) to unsaturated target expressions (t:$\mathcal{L}'_{T_i}$). Such expressions are later saturated and combined by a calculus semantics (see Section 7.8).

type
ENV = $\Sigma \xrightarrow[m]{} \mathcal{L}'_{T_i}$
$\mathcal{L}_S$,
$\mathcal{L}'_{T_i}$

There are no wellformed criteria for a semantics. All the semantic functions are thus complete. However, there are requirements to the semantics in order for it to be applicable in interpretation of an artefact model (see Section 7.7.1).

7.3.2 Semantics

Semantic environments are built through a parsing process which applies a set of semantic rules. These have the signatures:

value
$\mathcal{SM}$: $\mathcal{L}_S \to$ ENV $\to$ ENV,
$\mathcal{TM}$: **TM** $\to$ ENV $\to$ ENV,
$\mathcal{P}$: **P** $\to \Sigma \to \Sigma$,
$\mathcal{VS}$: **VS** $\to$ V-**set**,
$\mathcal{V}$: **V** $\to$ V

and the definitions:

$$
\begin{array}{rcl}
\mathcal{SM}[\![\text{template } tm]\!]\ \rho & = & \mathcal{TM}[\![tm]\!]\ \rho \\
\mathcal{SM}[\![sm_1; sm_2]\!] & = & \mathcal{SM}[\![sm_2]\!] \odot \mathcal{SM}[\![sm_1]\!] \\
\mathcal{TM}[\![(\ p\ ) : \{\ t\ \}]\!]\ \rho & = & \rho\,\dagger\,[\mathcal{P}[\![p]\!]\ [\,] \mapsto t] \\
\mathcal{TM}[\![tm_1, tm_2]\!] & = & \mathcal{TM}[\![tm_2]\!] \odot \mathcal{TM}[\![tm_1]\!] \\
\mathcal{P}[\![a\ =\ \{\ vs\ \}]\!]\ \sigma & = & \sigma\,\dagger\,[a \mapsto Seq(\mathcal{VS}[\![vs]\!])] \\
\mathcal{P}[\![a\ =\ [v_1; v_2]]\!]\ \sigma & = & \sigma\,\dagger\,[a \mapsto Intv(\mathcal{V}[\![v_1]\!], \mathcal{V}[\![v_2]\!])] \\
\mathcal{P}[\![p_1\ ,\ p_2]\!] & = & \mathcal{P}[\![p_2]\!] \odot \mathcal{P}[\![p_1]\!] \\
\mathcal{VS}[\![v]\!] & = & \mathcal{V}[\![v]\!] \\
\mathcal{VS}[\![vs_1, vs_2]\!] & = & \mathcal{VS}[\![vs_1]\!]\ \cup\ \mathcal{VS}[\![vs_2]\!] \\
\mathcal{V}[\![n]\!] & = & \text{Number}(n) \\
\mathcal{V}[\![\text{inf}]\!] & = & \text{Inf} \\
\mathcal{V}[\![\text{neginf}]\!] & = & \text{NegInf} \\
\mathcal{V}[\![v_p]\!] & = & \text{Name}(v_p)
\end{array}
$$

where we, in addition, overload $\odot$ for (ρ:ENV):

value
  $\odot$: (ENV → ENV) × (ENV → ENV) → (ENV → ENV)
  f $\odot$ g ≡ λ ρ:ENV • g(f(ρ))

7.4 Saturated and unsaturated targets

We mentioned in the introduction that there are two sorts of target expressions: *saturated* and *unsaturated*. By an unsaturated target expression, we understand a target expression which contains placeholders and unevaluated expressions. The placeholders are names which are not part of the target language for saturated targets. However, they stand for expressions which should be valid in the target language. Placeholders can be removed by substituting them with the proper target expressions they represent. E.g. such expressions can be atomic expressions like values or names valid in the target language.

The process is called *saturation* and is performed by a mixture of term–rewriting and evaluation of expressions. We have already seen examples of both saturated and unsaturated target expressions for a small sub–set of AutoLISP.

We can compare the distinction between saturated and unsaturated expressions with the distinction between ground terms and meta–variables in logic. The ground terms could be expressions built from operations like $\wedge$, $\vee$, $\rightarrow$, $\neg$ and constants true and false. Meta–variables can be introduced to represent expressions in ground terms and/or meta–variables. Expressions which contain meta–variables need to be saturated before evaluation to true or false can be performed.

The notion of saturation, as applied in this paper, has been inspired by Frege [71]. In *Function and Concept*, Frege makes a distinction between concepts (being functions returning truth values) and objects [71]. Concepts are unsaturated — they need objects in order to be saturated, just as functions need argument values in order to yield a result. The result of applying objects to concepts is a truth–value which states whether the object falls under the concept.

Today, we know that Frege's distinction was wrong. The higher–order logic of Church shows that functions (and thus also concepts) can have arguments and results which themselves are functions [46]. The use of the notion of saturation is in this paper just inspired by Frege's notion of concepts as functions. It has no real connection to the original meaning.

7.5 Subsumption of properties

In order to compare sets of properties of objects with property patterns of templates in a semantics, we need a notion of *subsumption* $<:$.

Definition 7.4 (Subsumption) Order–theoretically as in object–orientation, *subsumption* $<:$ is defined as (following [1]):

$$(\mathrm{p}{:}\mathrm{P}_1 \wedge (\mathrm{P}_1 <: \mathrm{P}_2)) \Rightarrow \mathrm{p}{:}\mathrm{P}_2$$

□

Considering types as sets of values, ":" corresponds to set–membership $\in$, and $<:$ to subset $\subseteq$. We shall apply subsumption for properties based on the intuition that if *x is an F* and whatever is *F is G*, then *x is a G* as well; where *F* and *G* are concepts constituted by sets of properties. Ontologically, we understand subsumption of property sets such that $ps_1 <: ps_2$ means that objects ascribed ps_2 are more specialised than objects ascribed ps_1. The intuition behind specialisation is that ps_2 includes more properties than ps_1. Thus ps_2 subsumes ps_1.

Increasing a set of properties makes the set of objects having all these properties decrease (or maintain the same). Thus, the order of sets of objects sharing common properties is dual to the order of sets of the properties, as illustrated on Figure 7.5. By *dual*, we understand that the order is reversed.

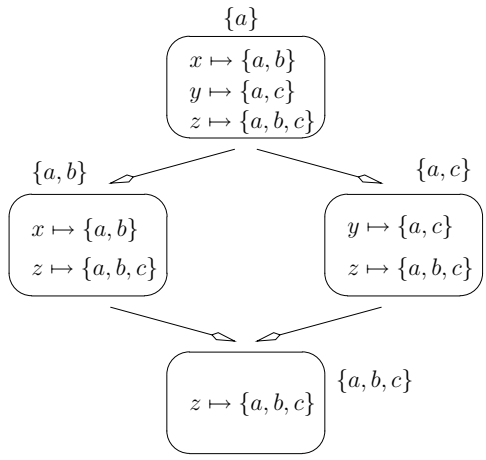


Figure 7.5: Duality of objects and common properties.

This is due to the Galois connection between objects and their common properties [85].

We need subsumption in two respects: (i) in determination of whether a given template complies with the properties ascribed a given object in an artefact model, and (ii) in defining an ordering of property patterns of templates such that the most specialised template can be selected.

Since we want subsumption to apply, not only to pairs of properties concerning their values, but also to property sets, we require that any binary relation composing two properties into a collection, satisfies *covariance* [1]:

Definition 7.5 (Covariance) A binary relation $A \circ B$ is *covariant* (in both arguments) if $A \circ B$ varies in the same sense as the arguments A and B:

$$A <: A' \wedge B <: B' \Rightarrow A \circ B <: A' \circ B' \tag{7.1}$$

□

Property sets expressed in $\mathcal{L}_M$ and property patterns expressed in $\mathcal{L}_S$ are specified using "," and ";" as separators. Semantically, they represent combination operations which obviously satisfy the criteria of covariance, as the model over which they range are maps and sets. In these cases, $\circ$ corresponds to the union set operation. The interpretation of expressions in $\mathcal{L}_M$ and $\mathcal{L}_S$ are designed such that covariance is satisfied.

In Chapter 4, Chapter 5, and Chapter 6, we modelled property values as possibly infinite sets. In this paper, we have two representations of value sets, namely intervals and sequences. Our definition of $<:$ complies with the set–theoretic intuition. It is defined hierachically over value sets, properties, and property sets (property patterns):

```
value
  <: : Σ × Σ → Bool,
  σ <: σ′ ≡
    (∀ a : A • a ∈ dom σ ⇒
      (a ∈ dom σ′ ∧ σ(a) <: σ′(a)))

  <: : (A × VS) × (A × VS) → Bool,
  (a,vs) <: (a′, vs′) ≡ a = a′ ∧ vs′ <: vs,

  <: : VS × VS → Bool
  vs <: vs′ ≡
    case (vs, vs′) of
      (Seq(vs), Seq(vs′)) → vs ⊆ vs′,
      (Seq(vs), Intv(v, v′)) →
        (∀ v″ : V • v″ ∈ vs ⇒ inintv(v″, Intv(v, v′))),
      (Intv(v, v′), Seq(vs)) →
        (∀ v″ : V • v″ ∈ vs ⇒ inintv(v″, Intv(v, v′))),
      (Intv(v, v′), Intv(v″, v‴)) →
        case ((v, v′), (v″, v‴)) of
          ((=NegInf, =Inf), (=NegInf, =Inf)) → true,
          ((=NegInf, =Inf), (Number(a′), =Inf)) → false,
          ((=NegInf, =Inf), (=NegInf, Number(b′))) → false,
          ((=NegInf, =Inf), (Number(a′), Number(b′))) → false,
          ((Number(a), =Inf), (=NegInf, =Inf)) → true,
          ((Number(a), =Inf), (Number(a′), =Inf)) →
            inintv(Number(a), Intv(v″, v‴)),
          ((Number(a), =Inf), (=NegInf, Number(b′))) →
            inintv(Number(a), Intv(v″, v‴)),
          ((Number(a), =Inf), (Number(a′), Number(b′))) →
            inintv(Number(a), Intv(v″, v‴)),
```

```
            ((=NegInf, Number(b)), (=NegInf, =Inf)) → true,
            ((=NegInf, Number(b)), (Number(a′), =Inf)) → false,
            ((=NegInf, Number(b)), (=NegInf, Number(b′))) →
              inintv(Number(b), Intv(v″, v‴)),
            ((=NegInf, Number(b)), (Number(a′), Number(b′))) → false,
            ((Number(a), Number(b)), (=NegInf, =Inf)) → true,
            ((Number(a), Number(b)), (Number(a′), =Inf)) →
              inintv(Number(a), Intv(v″, v‴)),
            ((Number(a), Number(b)), (=NegInf, Number(b′))) →
              inintv(Number(b), Intv(v″, v‴)),
            ((Number(a), Number(b)),(Number(a′), Number(b′))) →
              inintv(Number(a), Intv(v″, v‴)) ∧
              inintv(Number(b), Intv(v″, v‴))
        end
    end
  pre wf_VS(vs) ∧ wf_VS(vs′),
```

Interval membership ∈ is defined as follows:

```
value
  inintv: V × VS ~→ Bool
  inintv(v,vs) ≡
    case v of
      Number(n) →
        case vs of
          Intv(=NegInf,=Inf) → true,
          Intv(Number(a),=Inf) → a ≤ n,
          Intv(=NegInf,Number(b)) → n ≤ b,
          Intv(Number(a),Number(b)) → a ≤ n ∧ n ≤ b
          _ → false
        end,
      _ → false
    end
```

We assume the existence of binary relations overloaded for type Num.

7.6 Well–constrainedness

The language $\mathcal{L}_M$ is meant for modelling in an incremental way. Furthermore, it is meant for combining various models using, primarily, the lattice meet operation. Therefore, property values can be sequences or intervals. The principle is that the more narrow a property value set for an object is, the more precise is the design expression. Two values in a sequence means that either of the values are valid for the object.

When interpreting an artefact model it is, however, convenient that all attributes of objects in the model are ascribed single values only. In most cases, it is necessary.

Therefore, we apply three notions introduced in [78] concerning the space of interpretations of models. These are properties of design models in general:

over–constrained means that the set of possible interpretations of a model is empty because of too many (possibly contra dictionary) constraints to be satisfied. In our context, a model is *over–constrained* if an attribute of an object is ascribed the empty sequence $\perp$[10]. It may occur when lattice meet is applied on models which are not compliant; e.g. do not agree on the possible values for a properties of common objects.

under–constrained means that the set of possible interpretations of a model is too large; i.e. not feasible. In our context a model is *under–constrained* if some attributes of objects in a model are ascribed multiple values. In such cases, also multiple interpretations (given the same semantics) exist.

well–constrained means that the model is neither *over–constrained* nor *under–constrained*. In our context, it means that all attributes of objects in the model are ascribed single values.

Formally, we specify the three properties by the following predicates:

value
over_constrained: $\Theta \rightarrow$ **Bool**
over_constrained(θ) $\equiv$

[10] We do not consider ill–formed sequence specifications for property values to be subject to over–constrainedness because the three constraint predicates apply to well–formed model specifications. An ill–formed sequence specification like Intv(Inf,NegInf) results in a ill–formed model. In implementation, we represent ill–formed property values simply by the empty sequence $\perp$.

```
    (∃ x:X, a:A • x ∈ dom objects(θ) ∧ a ∈ dom fst(objects(θ)(x)) ∧
        fst(objects(θ)(x))(a)=⊥),

  under_constrained: Θ → Bool
  under_constrained(θ) ≡
    (∃ x:X, a:A, v:V, vs:V-set • x ∈ dom objects(θ) ∧
      a ∈ dom fst(objects(θ)(x)) ∧
      fst(objects(θ)(x))(a)=Seq(vs) ∧ card vs > 1),

  well_constrained: Θ → Bool
  well_constrained(θ) ≡
    ∼over_constrained(θ) ∧ ∼under_constrained(θ)
```

We require that an artefact model is *well–constrained* for proper interpretation to be made.

7.7 Properties of semantics

We specify a number of properties for our sets of a semantics. Such properties are important when considering interpretation of artefact models according to the semantics.

7.7.1 Completeness

We specify a criterion for the applicability of a semantics in interpretation of an artefact model. We say that the semantics is *complete* concerning that model. However, this notion of completeness is not to be confused with the notion of completeness known from the discipline of programming language semantics [161]. The distinction is that we are considering the relation between a semantics and an artefact model to be interpreted according to this semantics; not the properties of a inference system of the semantics.

Definition 7.6 (Complete semantics) A semantics represented by (e:ENV) is complete concerning a model represented by (θ:Θ) if and only if for all objects (x:X) designated in the model, there is a property pattern (σ:Σ) which is subsumed by the property set of x.

value

```
complete: ENV → Θ → Bool
complete(ρ)(θ) ≡
  (∀ x:X • x ∈ dom objects(θ) ⇒
    (∃ σ:Σ • σ ∈ dom ρ ∧
      σ <: fst(objects(θ)(x))))
```

□

We can easily ensure that a semantic environment is complete for all artefact models, by introducing the empty property pattern ⊤. If a semantic environment contains ⊤, we say that it is *bounded* upwards:

```
value
  ⊤ : Σ = mk_Σ([ ])

  bounded: ENV → Bool
  bounded(ρ) ≡ ⊤ ∈ dom ρ
```

However, the definition does not say what happens if ambiguity occurs.

7.7.2 Non–ambiguity

The notion of subsumption for patterns implies a partial order of these. In such an ordering, ambiguity can occur if two patterns share properties but are distinct and not related by subsumption. In such cases, it cannot be determined which pattern to chose for objects complying with both patterns. We thus require the existence of *lattice meet* of two such patterns. That is:

```
value
  non_ambiguous: ENV → Bool
  non_ambiguous(ρ) ≡
    (∀ σ, σ′:Σ • {σ,σ′} ⊆ dom ρ ∧
      σ ≠ σ′ ∧ ∼(σ <: σ′ ∨ σ′ <: σ) ⇒
        (∃ σ″:Σ • σ″ ∈ dom ρ ∧ σ″ = meet(σ,σ′)))
```

However, we do not require lattice meet to exist for all pairs. The structure we have here is a partial ordering with unique meet for all node pairs and

unambiguous join for the node pairs for which join is present. This structure is also called a *semilattice* [13].

7.8 Target saturation by term rewriting

Unsaturated target expressions, specified by templates in a semantics, are saturated by means of a calculus semantics which incorporates a set of rewriting rules for substitution and evaluation of expressions. There may exists distinct calculus semantics for distinct target expressions.

A semantic function which defines such substitution, evaluation, and rewriting, is one which takes an unsaturated target expression, an object environment and an object in that environment. In addition, it takes an operation which combines two target expressions. The result is a saturated target expression in which attributes are substituted with their corresponding property values for the given object, and in which expressions are evaluated and combined.

The unsaturated target language $\mathcal{L}'_{T_i}$ concerning only boxes and lines (like a small subset of AutoLISP) has the syntax:

$$
\begin{array}{rcl}
cs & ::= & (\ \mathsf{command}\ cmd\ ) \\
 & | & cs_1\ cs_2 \\
cmd & ::= & \text{"line"}\ l_1\ l_2\ \text{""} \\
 & | & \text{"rectangle"}\ l_1\ l_2 \\
 & | & cmd_1\ cmd_2 \\
l & ::= & (\ \mathsf{list}\ es\ ) \\
es & ::= & n \\
 & | & a \\
 & | & (\ e\ ) \\
 & | & es_1\ es_2 \\
e & ::= & +\ f_1\ f_2 \\
 & | & -\ f_1\ f_2 \\
 & | & *\ f_1\ f_2 \\
 & | & /\ f_1\ f_2 \\
 & | & (\ e\ ) \\
 & | & n \\
 & | & a
\end{array}
$$

where n range over numerals and a range over attributes names. Note, that the language is pre–fix concerning arithmetic operations. As in AutoLISP, there is a distinction between unary and binary minus. In AutoLISP, the distinction is based on whether there is a blank space after the operation. For unary minus there is not, which is why we treat unary minus at the scanning stage, and not as part of the semantics. Thus, n may stand for negative numbers as well.

We define a collection of semantic functions for saturating expressions in $\mathcal{L}'_{T_i}$. These have the signatures:

value
$\mathcal{CS}$: $\mathcal{L}'_{T_i} \to (\Theta \times \mathrm{X}) \to (\mathcal{L}_{T_i} \times \mathcal{L}_{T_i} \to \mathcal{L}_{T_i}) \xrightarrow{\sim} \mathcal{L}_{T_i}$,
$\mathcal{CMD}$: $\mathcal{L}'_{T_i} \to (\Theta \times \mathrm{X}) \xrightarrow{\sim} \mathcal{L}_{T_i}$,
$\mathcal{L}$: $\mathcal{L}'_{T_i} \to (\Theta \times \mathrm{X}) \xrightarrow{\sim} \mathcal{L}_{T_i}$,
$\mathcal{ES}$: $\mathcal{L}'_{T_i} \to (\Theta \times \mathrm{X}) \xrightarrow{\sim} \mathcal{L}_{T_i}$,
$\mathcal{E}$: $\mathcal{L}'_{T_i} \to (\Theta \times \mathrm{X}) \xrightarrow{\sim} \mathcal{L}_{T_i}$,
$\mathcal{F}$: $\mathcal{L}'_{T_i} \to (\Theta \times \mathrm{X}) \xrightarrow{\sim} \mathcal{L}_{T_i}$,

and the definitions:

$$
\begin{aligned}
\mathcal{CS}[\![\ \text{(command cmd)}\ ]\!]\ (\theta,x)\ \oplus &= \text{(command } \mathcal{CMD}[\![\ \text{cmd}\ ]\!]\ (\theta,x)\ \text{)}\\
\mathcal{CS}[\![\ \text{cs}_1\ \text{cs}_2\ ]\!]\ (\theta,x)\ \oplus &= (\mathcal{CS}[\![\ \text{cs}_1\ ]\!]\ (\theta,x)\ \oplus)\oplus\\
&\quad\ (\mathcal{CS}[\![\ \text{cs}_2\ ]\!]\ (\theta,x)\ \oplus)\\
\mathcal{CMD}[\![\ \text{"line" l}_1\ \text{l}_2\ \text{""}\ ]\!]\ (\theta,x) &= \text{"line" } (\mathcal{L}[\![\ \text{l}_1\ ]\!]\ (\theta,x))\\
&\quad\ (\mathcal{L}[\![\ \text{l}_2\ ]\!]\ (\theta,x))\ \text{""}\\
\mathcal{CMD}[\![\ \text{"rectangle" l}_1\ \text{l}_2\ \text{""}\ ]\!]\ (\theta,x) &= \text{"rectangle" } (\mathcal{L}[\![\ \text{l}_1\ ]\!]\ (\theta,x))\\
&\quad\ (\mathcal{L}[\![\ \text{l}_2\ ]\!]\ (\theta,x))\\
\mathcal{L}[\![\ \text{(list es)}\ ]\!]\ (\theta,x) &= \text{(list } (\mathcal{ES}[\![\ \text{es}\ ]\!]\ (\theta,x)\ )\ )\\
\mathcal{ES}[\![\ \text{(e)}\ ]\!]\ (\theta,x) &= \text{(} (\mathcal{E}[\![\ \text{e}\ ]\!]\ (\theta,x))\ \text{)}\\
\mathcal{ES}[\![\ \text{n}\ ]\!]\ (\theta,x) &= \Re(n)\\
\mathcal{ES}[\![\ \text{a}\ ]\!]\ (\theta,x) &= \Re(\text{fst}(\text{objects}(\theta)(x))(a))\\
\mathcal{ES}[\![\ \text{es}_1\ \text{es}_2\ ]\!]\ (\theta,x) &= \mathcal{ES}[\![\ \text{es}_1\ ]\!]\mathcal{ES}[\![\ \text{es}_2\ ]\!]\\
\mathcal{E}[\![\ +\ f_1\ f_2\ ]\!]\ (\theta,x) &= \text{ADD}((\mathcal{F}[\![\ \text{f}_1\ ]\!]\ (\theta,x)),\\
&\quad\ (\mathcal{F}[\![\ \text{f}_2\ ]\!]\ (\theta,x)))\\
\mathcal{E}[\![\ -\ f_1\ f_2\ ]\!]\ (\theta,x) &= \text{SUB}((\mathcal{F}[\![\ \text{f}_1\ ]\!]\ (\theta,x)),\\
&\quad\ (\mathcal{F}[\![\ \text{f}_2\ ]\!]\ (\theta,x)))\\
\mathcal{E}[\![\ *\ f_1\ f_2\ ]\!]\ (\theta,x) &= \text{MULT}((\mathcal{F}[\![\ \text{f}_1\ ]\!]\ (\theta,x)),\\
&\quad\ (\mathcal{F}[\![\ \text{f}_2\ ]\!]\ (\theta,x)))\\
\mathcal{E}[\![\ /\ f_1\ f_2\ ]\!]\ (\theta,x) &= \text{DIV}((\mathcal{F}[\![\ \text{f}_1\ ]\!]\ (\theta,x)),\\
&\quad\ (\mathcal{F}[\![\ \text{f}_2\ ]\!]\ (\theta,x)))\\
\mathcal{F}[\![\ \text{(e)}\ ]\!]\ (\theta,x) &= \text{(} (\mathcal{E}[\![\ \text{e}\ ]\!]\ (\theta,x))\ \text{)}\\
\mathcal{F}[\![\ n\ ]\!]\ (\theta,x) &= \Re(n)\\
\mathcal{F}[\![\ a\ ]\!]\ (\theta,x) &= \Re(\text{fst}(\text{objects}(\theta)(x))(a))
\end{aligned}
$$

Note, that each time a completely new target language is invented, the design system may require definitions of the underlying calculus. A fixed set of semantic functions limits to a fixed set of target languages.

Furthermore, $\Re$ and $\mho$ are dual operations of which $\Re$ encodes internal representations of property values (vs:VS) in $\mathcal{L}_{T_i}$, and $\mho$ gives the value which corresponds to such an encoding:

value
$\Re$: VS $\xrightarrow{\sim}$ VS_T
$\Re$(vs) **as** vs_T
pre wf_VS(vs)
post true,

$\mho$: $VS_T \rightarrow$ VS
$\mho(vs_T)$ **as** vs
post true

axiom $\forall$ n:VS, vs_T:
$\Re(\mho(n)) \equiv n$

The functions *ADD*, *SUB*, *MULT*, and *DIV*, perform the corresponding algebraic operations based on their target language arguments.

value
ADD: $\mathcal{L}_{T_i} \times \mathcal{L}_{T_i} \rightarrow \mathcal{L}_{T_i}$
ADD(n_1,n_2) $\equiv \mho(n_1) + \mho(n_2)$,

SUB: $\mathcal{L}_{T_i} \times \mathcal{L}_{T_i} \rightarrow \mathcal{L}_{T_i}$
SUB(n_1,n_2) $\equiv \mho(n_1) - \mho(n_2)$,

MULT: $\mathcal{L}_{T_i} \times \mathcal{L}_{T_i} \rightarrow \mathcal{L}_{T_i}$
MULT(n_1,n_2) $\equiv \mho(n_1) * \mho(n_2)$,

DIV: $\mathcal{L}_{T_i} \times \mathcal{L}_{T_i} \rightarrow \mathcal{L}_{T_i}$
DIV(n_1,n_2) $\equiv \mho(n_1) \ / \ \mho(n_2)$,

The operation $\oplus$ in this context can simply be string appending.

In the considered target language, we only allow numerals as representations for property values. This defines a restriction of the range of artefact models possible to interpret into targets. However, other target languages may allow for such property value representations.

7.9 Interpretation of artefact models

We have now presented the mechanisms for performing full interpretation of artefact models. The result is a view expressed in a target language $\mathcal{L}_T$. This view is the result of interpreting each object in a model according to the semantics. This is done by selecting a property pattern in the domain of the semantic environment. This pattern has to comply with the properties ascribed the object. The result is an unsaturated target expression for that object. The target expression is saturated by means of the calculus semantics. The target expressions, representing the meaning of objects in the artefact model, are finally combined.

Figure 7.6 depicts the principle of semantic parameterized interpretation. The figure gives a detailed description of the principles, compared to Figure 7.3.

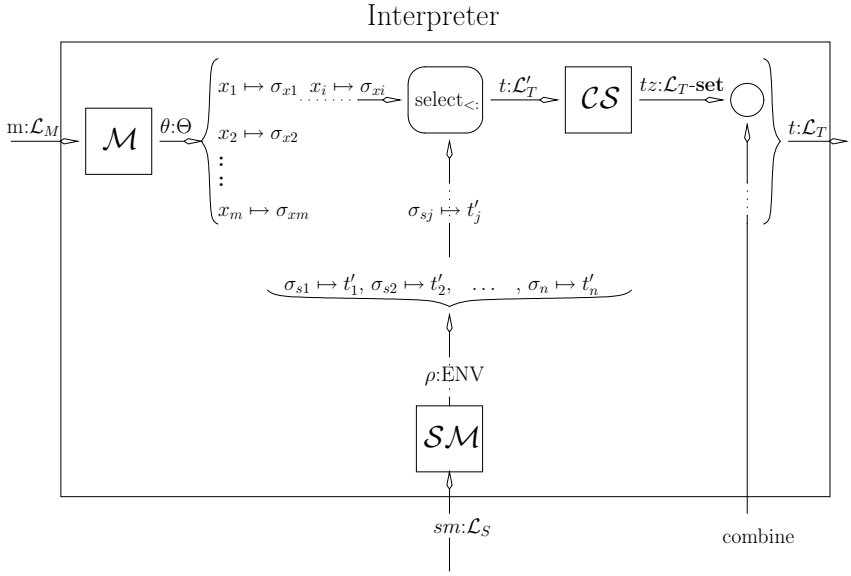


Figure 7.6: Detailed picture of semantic parameterized interpretation.

type
$\mathrm{CS} = \mathcal{L}'_{T_i} \to (\Theta \times \mathrm{X}) \to (\mathcal{L}_{T_i} \times \mathcal{L}_{T_i} \to \mathcal{L}_{T_i}) \xrightarrow{\sim} \mathcal{L}_{T_i}$

value
$\mathcal{I}'$: $\Theta \to \mathrm{ENV} \to \mathrm{CS} \to (\mathcal{L}_{T_i} \times \mathcal{L}_{T_i} \to \mathcal{L}_{T_i}) \xrightarrow{\sim} \mathcal{L}_{T_i}$**-set**
$\mathcal{I}'(\theta)(\rho)(\mathcal{CS})(\oplus) \equiv$

```
    {t|t:L_Ti •
       (∃ x:X • x ∈ dom objects(θ) ∧
          let t′=select(fst(objects(θ)(x)),ρ) in
             t=CS⟦ t′ ⟧ (θ,x) ⊕
          end)}
  pre complete(ρ)(θ) ∧ well_constrained(θ) ∧
     non_ambiguous(ρ) ∧ bounded(ρ)
```

The function select[11] is defined as follows:

```
value
  select: Σ × ENV ~→ L′_Ti
  select(σ_limit,ρ) as t
  post let σ:Σ • σ ∈ dom ρ ∧
            most_specialised(σ,ρ)(σ_limit) in
           t=ρ(σ)
       end
  pre bounded(ρ),

  most_specialised: Σ × ENV → Σ ~→ Bool
  most_specialised(σ,ρ)(σ_limit) ≡
     σ <: σ_limit ∧
     (∀ σ′:Σ • σ′ ∈ dom ρ ∧ σ′ <: σ_limit ⇒ σ′ <: σ)
  pre σ ∈ dom ρ
```

Full interpretation of an artefact model (m:$\mathcal{L}_M$) given a semantics (s:$\mathcal{L}_S$) and a calculus semantics ($\mathcal{CS}$:CS), is now defined as:

```
value
  I: L_M → L_S → CS → (L_Ti × L_Ti → L_Ti) → L_Ti
  I(m)(sm)(CS)(⊕) ≡
     combine(I′(M⟦ m ⟧)(SM⟦ sm ⟧([ ]))(CS)(⊕))
```

where combine has the signature:

```
value
  combine: L_Ti-set → L_Ti
```

[11] Which selects a proper entry in (ρ:ENV) such that it "best fits" σ:Σ.

The function combine could be similar to a folding using the $\oplus$ operation from Section 7.8.

It is important to note that the presented interpretation algorithm and principle is designed for target languages which are compositional. That is, the meaning of an artefact model is a function of the meaning of its objects. This excludes views which require that the meaning of objects are calculated from the meanings of other objects (in a non–cyclic way). However, in most normal cases, it is often possible to write expressions in some functional languages such that the views can still be defined in a compositional way.

By that we understand that the meaning of objects can be determined independently and composed in a final step. There may be target languages for which such an interpretation principle does not apply. Also, including relations in artefact models in the interpretation requires a similar way of determining the meaning of objects based on the meaning of other objects. The latter, we discuss further in Section 7.10.1.

7.10 Conclusion

In this paper, we have suggested a new software architecture for conceptual design systems. The idea for the architecture has arised from the observation that today's design systems lack of dynamics with respect to incremental specialisation of design objects and with respect to dynamic evolution of type systems for such objects.

The architecture presented aims at introducing these dynamics by means of what we call *semantic parameterised interpretation*. In systems based on this concept, design models (artefact models) are written in a formal language $\mathcal{L}_M$. In $\mathcal{L}_M$, we are able to introduce objects, ascribe properties to objects, and add relations between objects. Furthermore, $\mathcal{L}_M$ offers two operations *join* and *meet* which are binary operations on artefact models and which combine these according to consistent laws. We have specified the syntax and formal semantics of $\mathcal{L}_M$.

Design models refer to names of properties, relations, and values of these. The meanings of these names are not statically defined as in object–oriented class systems. The meaning of the names are given by a semantics which is written in a formal language $\mathcal{L}_S$. Thereby, we are able to introduce new names for properties, relations, and values, by defining the meaning of these names in the semantics. Thus, introducing such new names do not require a large amount of

programming nor rearrangement nor recompilation of the program code of the design system. We have specified the syntax and formal semantics of $\mathcal{L}_S$.

Artefact models are now interpreted according to a semantics and a calculus for evaluating expressions. The result is a *view* of the model which is a written specification in some target language $\mathcal{L}_T$. It is essential to our principle that the same artefact model can be subject to many different interpretations — a principle which is highly important in today's distributed and many–sorted realm of construction. Thus we have shown how we can derive views for graphical presentation in AutoCAD, weight calculations, and stress calculations using MatLab.

The architecture is based on the notion of property–orientation in which the meaning of an object is the set of its properties. In order to be able to distinguish objects with equivalent properties, object identifiers has been introduced.

The interpretation of models is based on the properties of the objects, such that objects with the same properties have the same meaning. However, this induces problems if objects are described by the same set of properties even though they are intended to be conceptually distinct. We have argued that often it is convenient to introduce a property which simply states the *sort* of object at hand. This property, in a sense corresponds to the rôle played by class names in object–oriented programming languages. Thereby, we are able to make a distinction between, e.g. the plate for a door and the plate for a table, even though these two kinds of objects may be equivalent with respect to dimensions and material. Even though, the solution does not break our principle of property–orientation, but it *does* shows the rationality in sometimes making a distinction between class names and the types they denote. The solution is, however, flexible as stating a *sort* property is optional; we can even call the attribute something else or omit it if it is not needed.

A design tool with the presented architecture is generic in the sense that the semantics — as a parameter — determines the sorts of entities that can be designed. Thus, a semantics *specialises* the architecture into a certain application. Thereby it has similarities with the principles of *compiler generators* — a principle and theory which has inspired this paper.

It may be argued that the problem we have tried to solve is much easier solved by means of database schemes. In such schemes, objects can be introduced as rows and properties of objects can be introduced as columns. There are, however, two main reasons why a database approach is not satisfactory. First, dynamically changing the attributes of a database schema — known as *schema evolution* — is in general complex and often problematic. Second, the result of database operations and queries are either database schemas, tuples, or the

field values of relations. It is essential to our approach that we are able to define the meaning of artefact models as expressions in many different incomparable target languages.

We believe that the introduction of semantic parameterized interpretation can serve as inspiration for future research in this area, and that we — by our philosophical and design concerned study — has emphasized important issues and solutions relevant to design practitioners and software industry working in the interdisciplinary field of design and IT.

7.10.1 Future work: handling relations

Future work may focus on clarifying various issues. One is the compositionality principle in the interpretation functions which limit the ways interpretation can be made. Several sorts of views may require that the meaning of individual objects or some of their properties are combined in a non–compositional way. Another issue concerns the notion of relations. In the following, we shall explore both issues but with an emphasis on the latter.

Until now, we have ignored relations in interpretation artefact models. The modelling language $\mathcal{L}_M$ has been designed on an ontological foundation which makes a distinction between two sorts. In design, such states of affairs are the configuration of objects. The entities are properties and relations. Often for physical systems like buildings, properties are categorised into *intrinsic* and *extrinsic* [64, 40, 38]. By an *intrinsic property*, we understand a property which exists independently of any other object or surroundings. By an *extrinsic property*, we understand a property which an object has by virtue of another object; i.e. the existence of an extrinsic property depends on the existence of another object than the object possessing the extrinsic property.

Ontologically, following Shoemaker [153], we believe extrinsic properties to be less compliant with the mental process of designing than those of relations. E.g. we may wish to express that two objects are to be two feet apart. This information can easily be expressed as a topological relation, but it can also (as common practice in object–oriented design systems) be expressed by introducing a coordinate system and expressing position and orientation as properties of the objects. The former approach, we believe is closest to the mental process of designing. In addition, it makes deduction of knowledge more simple as rules can be defined, e.g. as Prolog programs. The latter approach is more efficient computationally when it comes to visualisation. Such visualisations utilize the notion of coordinate systems on paper or screen.

A conceptual design tool should, we claim, favour the former approach. Therefore, we have designed $\mathcal{L}_M$ for models to express a clear distinction between intrinsic properties of objects and relations between objects. However, many target languages are rooted in the latter approach.

The principle of semantic parameterized interpretation is based on a compositional semantics. This principle is convenient for handling intrinsic properties, as the meaning of each object can be computed for later to be combined with the meaning of other objects.. Handling relations, following the latter approach, requires a different interpretation approaches, as the meanings of each object may depend on the meaning of other objects. In essence, a way of binding free terms such that all relations hold, needs to be incorporated in a non–compositional way.

For topological relations this problem is well known within the area of *field science*. The problem is that of defining targets when the information is inadequate for saturating all free terms, and of finding a substitution such that all relations hold. The problem takes the form as a combinatorial problem. Consider the following model which makes use of three topological relations: *xrel* means that the two objects are translated a certain distance from each other, *xzrot* means that the objects are rotated a certain angle in the xz–plane according to each other, and *yzrot* means that the objects are rotated a certain angle in the yz–plane according to each other.

```
model
  w1 :  (sort ={wall},
         length={1800},
         height={600},
         width={800});
  w2 :  (sort={wall},
         length={1800},
         height={600},
         width={800});
  (w1,w2) :(xrel={900}, xzrot={0}, yz={0})
```

The solution space, which satisfies the relations between the two objects, may be infinite as both objects can be placed in infinite many ways and still satisfying the relation. Furthermore, this solution space lies in a much larger space which is the space of interpretations which do not satisfy the relations of the artefact model. The difference in size between the two spaces makes finding suitable solutions a hard task to solve computationally [125].

Figure 7.7 shows that there are many possible ways of positioning the two walls in the plane. Fixing the position and orientation of the object $w1$ gives the possibilities of placing the object $w2$ anywhere in a distance of 900 units as well

as rotated in the xy–plane.

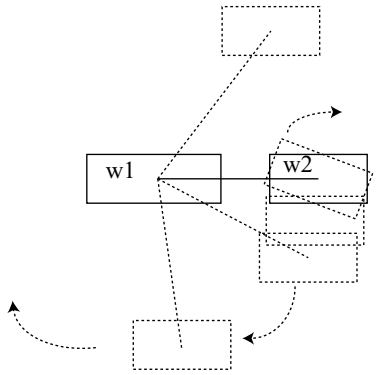


Figure 7.7: Possible ways of positioning the walls.

The space of possible interpretations is called the solution space. This space may be infinite when relations like topological relations are introduced in artefact models. Often, only a small sub–space of the solution space should be valid, as much design knowledge may not be expressed, thereby creating a large solution space.

One way to solve the problem is to encode topological information as properties. This, we believe, does not break our ontological understanding as we are talking about representation. The encoding process can be done automatically for topological relations and could be part of the preparation for the semantic parameterized interpretation. However, the encoding process may need algorithms for solving constraint satisfactions of relations and for making optimisation. Still, these are known issues so we leave it to future work.

Another way to solve the problem is simply to leave the task of placing objects topologically to the visualisation tool. In a design tool like AutoCAD, the individual objects can be positioned in an interactive process with the user. Relation constraints can then be added by selecting pairs of objects and specifying their relations. This solution leaves more to the visualisation tool but without violating our original motivations.

7.10.2 Disclaiming a seemingly risky business

It can be argued that founding a design system on the ability to specify the meanings of constructs in conceptual design models, is a risky business. However, at some level this is what we actually do in most cases. The programmer

of a CAD application writes procedures for visualization and manipulation of objects, in requirements engineering tables and definitions formally as well as informally specify the meaning of words and modelling constructs, etc. An example of the latter is a table which informally defines the meaning of constructs in a graphical design notation. Another example is a set of validation formulae for defining whether a design fulfills requirements. The latter exemplifies a situation in which messing with the semantics can cause serious damage. We should thus state that the intention of having a language $\mathcal{L}_S$ is not for loosening the constraints and framework for validation and verification. Note, that it is a semantics defining views on a model. Basically, we are all interested in having meaningful views. Also, introducing $\mathcal{L}_S$ is primarily for the purpose of investigating the foundation for conceptual design systems and their application.

Furthermore, although we show how semantic flexibility can be incorporated, it does not imply that everybody should have access to this feature. There may here be two extremes: (i) The semantics is fixed and cannot be touched by the designer/user, and (ii) the designer/user can manipulate the semantics.

The former extreme can be extended by having a set of semantics; i.e. adding an additional parameter to the interpreting process, for stating the desired one.

In between the two extremes, there are the following possibilities:

- We have a root semantics from which we can make copies that can be edited. Certain satisfiability constraints are defined for all derived semantic specifications.
- Certain parts of a semantics are accessible for editing; others are not.
- etc.

The second possibility corresponds to parameterized modelling.

Basically, the issue addressed is reduced to "*who has editorial access to the semantics*?" This, however, is a discussion separate from that of the relation between models and their meanings.

We shall not discuss the above issues further in the paper; they are simply a disclaimer for this paper.

7.11 Appendix A: Concrete syntax of $\mathcal{L}_M$

The following is the concrete syntax of $\mathcal{L}_M$.

```
   W  ::=  M Wopt
Wopt  ::=  ⊓ M Wopt | ⊔ M Wopt | ϵ
   M  ::=  "model" S Sopt
   S  ::=  id ":" "(" P Popt ")" D
      |    "(" id "," id ")" ":" "(" P Popt ")"
Sopt  ::=  ";" S Sopt | ϵ
   D  ::=  "{" id Dopt "}" | ϵ
Dopt  ::=  "," id Dopt | ϵ
   P  ::=  id "=" Q Popt
Popt  ::=  "," P Popt | ϵ
   Q  ::=  id | { V VS } | "[" V; V "]"
  VS  ::=  "," V VS | ϵ
   V  ::=  n | id | inf | neginf
```

7.12 Appendix B: Concrete syntax of $\mathcal{L}_S$

The following is the concrete syntax of $\mathcal{L}_S$.

```
   SS  ::=  SM SSopt
SSopt  ::=  ";" SM SSopt | ϵ
   SM  ::=  template TM TMopt
   TM  ::=  "(" P Popt ")" ":" "{" T "}"
TMopt  ::=  "," TM TMopt | ϵ
    P  ::=  id "=" Q Popt
 Popt  ::=  "," P Popt | ϵ
    Q  ::=  id | { V VS } | "[" V; V "]"
   VS  ::=  "," V VS | ϵ
    V  ::=  n | id | inf | neginf
```

T range over unsaturated target expressions.

7.13 Appendix C: Lattice operations

```
value
  join: Θ × Θ ~→ Θ
  meet: Θ × Θ ~→ Θ

axiom ∀ θ, θ′ : Θ •
  join(θ,θ′) ≡
    mk_Θ([ x↦(σ″,xs″) | x:X, σ″:Σ, xs″:X-set •
      x ∈ dom objects(θ) ∧ x ∈ dom objects(θ′) ∧
        σ″ = join(fst(objects(θ)(x)),fst(objects(θ′)(x))) ∧
          xs″ = join(snd(objects(θ)(x)),snd(objects(θ′)(x))) ],
      [ (x,x′)↦σ | x, x′:X, σ:Σ •
        (x,x′) ∈ dom relations(θ) ∧ (x,x′) ∈ dom relations(θ′) ∧
        σ = join(relations(θ)(x,x′),relations(θ′)) ]),

  meet(θ,θ′) ≡
    mk_Θ([ x↦(σ″,xs″) | x:X, σ″:Σ, xs″:X-set •
      (x ∈ dom objects(θ) ∧ x ∈ dom objects(θ′) ∧
       σ″ = meet(fst(objects(θ)(x)),fst(objects(θ′)(x))) ∧
         xs″ = snd(objects(θ)(x)) ∪ snd(objects(θ′)(x))) ∨
      (x ∈ dom objects(θ) ∧ x ∉ dom objects(θ′) ∧
       σ″ = objects(θ)(x)) ∨
      (x ∉ dom objects(θ) ∧ x ∈ dom objects(θ′) ∧
       σ″ = fst(objects(θ)(x))) ],
     [ (x,x′)↦ σ | x,x′:X, σ:Σ •
       ((x,x′) ∈ dom relations(θ) ∧ (x,x′) ∈ dom relations(θ) ∧
        σ″ = meet(relations(θ)(x,x′),relations(θ′))) ∨
       ((x,x′) ∈ dom relations(θ) ∧ (x,x′) ∉ dom relations(θ′) ∧
        σ″ = relations(θ)(x,x′)) ∨
       ((x,x′) ∉ dom relations(θ) ∧ (x,x′) ∈ dom relations(θ′) ∧
        σ″ = relations(θ′)) ])



value
  meet: Σ × Σ → Σ,
  join: Σ × Σ → Σ,

axiom ∀ σ, σ′:Σ •
  join(σ,σ′) ≡
    mk_Σ([ a↦vs|a:A, vs:VS •
      a ∈ dom σ ∧ a ∈ dom σ′ ∧ vs=join(σ(a),σ′(a)) ]),
```

```
  meet(σ,σ′) ≡
    mk_Σ([ a↦vs|a:A, vs:VS •
      (a ∈ dom σ ∧ a ∈ dom σ′ ∧ vs=meet(σ(a),σ′(a))) ∨
      (a ∈ dom σ ∧ a ∉ dom σ′ ∧ vs=σ(a)) ∨
      (a ∉ dom σ ∧ a ∈ dom σ′ ∧ vs=σ′(a)) ])

value
  max: V × V → V
  min: V × V → V

value
  ∩: (V × V) × (V × V) → VS
  ∩(intv,intv′) ≡
    case (intv,intv′) of
      ((NegInf,Inf),(NegInf,Inf)) → Intv(NegInf,Inf)
      ((NegInf,Inf),(NegInf,Number(b′))) → Intv(NegInf,Number(b′))
      ((NegInf,Inf),(Number(a′),Inf)) → Intv(Number(a′),Inf)
      ((NegInf,Inf),(Number(a′),Number(b′))) →
        if a′<b′ then Intv(Number(a′),Number(b′))
        elsif a′=b′ then Seq({Number(a′)})
        else ⊥
        end
      ((NegInf,Number(b)),(NegInf,Inf)) → Intv(NegInf,Number(b))
      ((NegInf,Number(b)),(NegInf,Number(b′))) →
        Intv(NegInf,Number(min(b,b′)))
      ((NegInf,Number(b)),(Number(a′),Inf)) →
        if a′<b then Intv(Number(a′),Number(b))
        elsif b=a′ then Seq({Number(b)})
        else ⊥
        end
      ((NegInf,Number(b)),(Number(a′),Number(b′))) →
        if a′<b then Intv(Number(a′),Number(min(b,b′)))
        elsif b=a′ then Seq({Number(b)})
        else ⊥
        end
      ((Number(a),Inf),(NegInf,Inf)) → Intv(Number(a),Inf)
      ((Number(a),Inf),(NegIng,Number(b′))) →
        if a<b′ then Intv(Number(a),Number(b′))
        elsif a=b′ then Seq({Number(a)})
        else ⊥
        end
```

```
        ((Number(a),Inf),(Number(a′),Inf)) →
            Intv(Number(max(a,a′)),Inf)
        ((Number(a),Inf),(Number(a′),Number(b′))) →
            if a<b′ then Intv(Number(max(a,a′)),Number(b′))
            elsif a=b′ then Seq({Number(a)})
            else ⊥
            end
        ((Number(a),Number(b)),(NegInf,Inf)) →
            Intv(Number(a),Number(b))
        ((Number(a),Number(b)),(NegInf,Number(b′))) →
            if a<b′ then Intv(Number(a),Number(min(b,b′)))
            elsif a=b′ then Seq({Number(a)})
            else ⊥
            end
        ((Number(a),Number(b)),(Number(a′),Inf)) →
            if a′<b then Intv(Number(max(a,a′)),Number(b))
            elsif a′=b then Seq({Number(a′)})
            else ⊥
            end
        ((Number(a),Number(b)),(Number(a′),Number(b′))) →
            if b=a′ then Seq({Number(b)})
            elsif b′=a then Seq({Number(b′)})
            elsif a′<b and also a<b′ then
                Intv(Number(max(a,a′)),Number(min(b,b′)))
            else ⊥
            end
        (_,_) → ⊥
    end

∪: (V × V) × (V × V) → VS
∪(intv,intv′) ≡
    case (intv,intv′) of
        ((NegInf,Inf),(NegInf,Inf)) → Intv(NegInf,Inf)
        ((NegInf,Inf),(NegInf,Number(b′))) → Intv(NegInf,Inf)
        ((NegInf,Inf),(Number(a′),Inf)) → Intv(NegInf,Inf)
        ((NegInf,Inf),(Number(a′),Number(b′))) → Intv(NegInf,Inf)
        ((NegInf,Number(b)),(NegInf,Inf)) → Intv(NegInf,Inf)
        ((NegInf,Number(b)),(NegInf,Number(b′))) →
            Intv(NegInf,Number(max(b,b′)))
        ((NegInf,Number(b)),(Number(a′),Inf)) → Intv(NegInf,Inf)
        ((NegInf,Number(b)),(Number(a′),Number(b′))) →
            Intv(NegInf,Number(max(b,b′)))
        ((Number(a),Inf),(NegInf,Inf)) → Intv(NegInf,Inf)
        ((Number(a),Inf),(NegIng,Number(b′))) → Intv(NegInf,Inf)
```

```
    ((Number(a),Inf),(Number(a′),Inf)) → Intv(Number(min(a,a′)),Inf)
    ((Number(a),Inf),(Number(a′),Number(b′))) →
      Intv(Number(min(a,a′)),Inf)
    ((Number(a),Number(b)),(NegInf,Inf)) → Intv(NegInf,Inf)
    ((Number(a),Number(b)),(NegInf,Number(b′))) →
      Intv(NegInf,Number(max(b,b′)))
    ((Number(a),Number(b)),(Number(a′),Inf)) →
      Intv(Number(min(a,a′)),Inf)
    ((Number(a),Number(b)),(Number(a′),Number(b′))) →
      Intv(Number(min(a,a′)),Number(max(b,b′)))
    (_,_) → ⊥
  end,

value
  ∈: V × (V × V) → Bool

axiom n, a, b:Num ≡
  Number(n) ∈ (NegInf,Inf) = true,
  Number(n) ∈ (NegInf,Number(b)) = n≤b,
  Number(n) ∈ (Number(a),Inf) = a≤n,
  Number(n) ∈ (Number(a),Number(b)) = (a<n ∧ n<b) ∨ a=n ∨ n=b,
  (_,_) → false

value
  join: VS × VS → VS,
  meet: VS × VS → VS

axiom •
  join(Seq(vs),Seq(vs′)) ≡ Seq(vs ∪ vs′),
  join(Seq(vs),Intv(v,v′)) ≡
    Seq({v|v:V • v ∈ vs ∨ v ∈ (v,v′)}),
  join(Intv(v,v′),Seq(vs)) ≡
    Seq({v″|v″:V • v″ ∈ vs ∨ v″ ∈ (v,v′)}),
  join(Intv(v,v′),Intv(v″,v‴)) ≡ Intv((v,v′) ∪ (v″,v‴)),

  meet(Seq(vs),Seq(vs′)) ≡ Seq(vs ∩ vs′),
  meet(Seq(vs),Intv(v,v′)) ≡
    Seq({v|v:V • v ∈ vs ∧ v ∈ (v,v′)}),
  meet(Intv(v,v′),Seq(vs)) ≡
    Seq({v″|v″:V • v″ ∈ vs ∧ v″ ∈ (v,v′)}),
  meet(Intv(v,v′),Intv(v″,v‴)) ≡ Intv((v,v′) ∩ (v″,v‴))
```

Part IV

Philosophy

Chapter 8

Object aspects

Abstract: References to objects like buildings, walls, and bricks, appear repeatingly in descriptions like requirements and design documents. However, the objects referred to may not have physical presence, which makes it difficult to claim that the descriptions are meaningful. Applying a formal approach to mereology reduces part–whole relations to a logical formalism and seems to avoid such problems, but the connection to reality is then weak. In this paper, we seek an argument for connecting object descriptions in language to a causal realm. However, claiming such a connection introduces a number of mereological problems. Of these, the problem of arbitrary sums is the most dominant. Basically, we cannot accept that an object does not fall under any concept. If we wish — as common in object references in language — to accept arbitrary sums of objects, we must also accept arbitrary concept and that we cannot. The paper, suggests the notion of object aspects as the entities referred to when referring to potential objects. An object aspect is a part existing in a possible world which is causally reachable from the present world. We defend our perspective against the major criticisms of extensional mereology. In the defence, we argue that object aspects can contribute to the meaning of descriptions of objects like buildings, walls, and bricks, without problematic ontological commitments.

8.1 Introduction

The relation between parts and wholes is often important ingredients in the identification of objects referred to in sentences and expressions. If I say "*the roof of the house*", I mean a certain object which I consider part of a certain house. The naming of the roof by means of the definite article "*the roof*", is one component in the identification of that object. The reference to the relation in which the roof stands to the house is another component. This relation restricts the interpretation space for objects falling under the concept roof.

Part–whole relations are important in a wide range of informal and formal descriptions of things; actual or possible. Such relations may be applied when talking about objects in requirements and design of artefacts like buildings, or when talking about the construction processes which build up such artefacts from collections of parts.

It is tempting to utilize a version of extensional mereology for doing so in formalisation or precisely formulated documents. "Extensional mereology" is a common term which Simons takes to denote formal theories of part–whole relations [155]. What prominent extensional mereological theories aim at covering is, however, more far reaching than what is often needed in order for sentences expressing part–whole information to make sense.

Extensional mereology can be seen as working on two levels: A logical level and an epistemological level. These are two different levels of abstraction.

The logical level concerns descriptive or prescriptive formulations which intend to catch a still picture of part–whole constellations of objects. Logical operations are defined algebraically, and logical names are taken to denote actual or imaginary objects. E.g. $x + y$ denotes the mereological sum of the objects denoted by x and y, and $x \leq z$ expresses that the object denoted by x is a part of the object denoted by z.

The epistemological level is deeper and more universal but also much more committing and problematic. It aims at linking part–whole relations at a logical level, to reality.

One of the two major critiques of extensional mereology is that it does not fit with reality or how objects are considered in every day life. This means that extensional mereology fails to establish the link.

In order to bridge the gap, I believe mereological discussions need to consider

the notion of truth–conditions. It has been shown in other metaphysical areas that the notion of truth–conditions has an important rôle to play when adding epistemological perspectives. Thus, it was used by Frege to approach a definition of the notion of sense, and it underlies much of the discussion on properties.

In a mereological discussion like the one in this paper, I believe, we need to consider the notion of truth–conditions for whether an object is part of a whole. Here, we are engaged in solving at least two problems. First, we need to offer an account for how to handle reference to a plurality of objects which are referred to as a whole. Second, we need to decide whether conceptually to distinguish objects which undergo small changes; and if so, how this should be done.

As an example of the first problem, consider the sentence: "*walls and ceilings are painted white*". Objects of two different kinds (walls and ceilings) are here predicated. The question is whether we should or can consider these objects as some whole being predicated. The problem is called *the problem of arbitrary sums*.

As an example of the second problem, consider the situation of describing the construction tasks to be performed on a concrete foundation. Anchors are mounted, small holes and drains are made, the foundation is primed two or three times, etc. At each stage, the description of a specific task may refer to the concrete foundation simply as "*the concrete foundation*" — not as "*the concrete foundation being primed twice*", etc. The question is now whether we need to commit ourselves to concepts which are abstractions of the concrete foundation in each stage of development, and of which the concrete foundation, at each stage, makes the extension of unique concepts. The problem is called *the problem of flux*.

Both problems are present in mereology on the epistemological level; only the second problem is present at the logical level of mereology. I believe, though, that remaining on the logical level is too shallow an approach — we need more means for justification of part–whole relations referred to in sentences and expressions.

The paper is structured as follows. In Section 8.2, I motivate the introduction of object aspects and further in Section 8.3, I argue for their existence in a causal realm. The argumentation is defended against two other views on reference: *Platonism* and *empiricism*. Section 8.4 and Section 8.5 defend the notion of object aspects against the two major charges against mereology: *The problem of arbitrary sums* and *the problem of flux*. The remaining sections are appendix in which I shall discuss some other charges against extensional mereology. These discussions indirectly relate to object aspects as they concern mereology in general.

8.2 Object aspects

In this paper, I introduce the notion of *object aspects*. It is a notion which I have found convenient when needing to refer to parts of potential objects, without fully committing myself to extensional mereology at the epistemological level. The notion of object aspects, as a concept of reference, lies between the complexities of the two mereological levels presented.

By an object aspect, I understand a part of an object existing in a *potential world*. A potential world is one which can be causally reached from the actual world. Objects existing in possible worlds are possible objects and likewise are objects existing in potential worlds considered potential objects. Object aspects are special potential objects in the sense that they are proper parts of potential objects. Thus, an object aspect exists by virtue of there being a corresponding part of an object in a potential world. The outer surface of an exterior wall exists as an object aspect because that wall can exist in a potential world and the surface is a part of that wall. Similar does a concrete foundation exists as an object aspect because there is a potential building of which it is a part. The relation between an object aspect and the whole of which it is an aspect, can be formally defined as:

$$[\![\ a \ \mathsf{aspect\text{-}of}\ b\ ]\!] \equiv \hat{\exists} w : \mathbf{W} \bullet w_0 \leq w \wedge a \in w \wedge b \in w \wedge a \ll_w b$$

where w_0 denotes the actual world, $\ll_w$ is the proper part relation holding in the world w, and $\in$ means world membership. The symbol $\hat{\exists}$ is taken to denote existential quantification over the set of possible worlds $\mathbf{W}$. A proper part — following Simons — is a part of an object such that there exists another part which does not coincide with the former and which is not the empty part [155].

The notion of *parts* in extensional mereology can be taken to fit various interpretations, and I shall thus distinguish it from the notion of object aspects for which I introduce for the reasons already mentioned. Thus, the notion of object aspects is one specific interpretation of the notion of *parts*, although I believe that a distinction between the two is convenient to maintain.

I believe that references to object aspects can contribute to the meaning of sentences and expressions about objects like artefacts, and that its more modest ontological commitments makes it avoid certain mereological problems. However, the notion is not aimed at capturing the deepest epistemological issues; nor is it intended to be as general as that of *parts*, because its purpose is more narrow.

Obviously, a theory of object aspects strongly relates to a theory of parts. However, in comparison, the notion of object aspects is broader in one sense and

more narrow in another.

It is broader in the sense that object aspects abstract from physicality, time, and causal connections, and in the sense that I shall allow for what *seemingly* is reference to *arbitrary mereological sums*.

The notion is more narrow in the sense that the objects of which we have aspects are restricted to what we usually understand by "things" like buildings, furniture, books, etc. A more correct term is "*continuants*" which, according to Simons, is defined by Broad as objects which do not have temporal parts [155] like events and processes. Objects which do (called "*occurrents*"), we exclude altogether in this treatment.

Furthermore, the notion of object aspects is more narrow, as object aspects are parts of objects in potential worlds; not in all possible worlds.

To some extent, I follow Lewis' notion of possible worlds, according to which some worlds are causally connected and others are counterfactual to each other [120, 119].

Also, I take object aspects to be *contextual* in the sense that their existence depend on the existence of wholes of which they are parts (even though these wholes are only potential). Such a whole restricts the interpretation space of parts we consider to be aspects of the whole. Consider a ball of solid metal. The ball considered in isolation is an object but in this context not an object aspect as it is not a proper part of anything. We can refer to the ball and its properties, but we do not require any additional knowledge related to its surroundings for identification. However, if the ball is placed as part of a ball–bearing device, it is a part of that device. In that case, we may need to refer to the *ball–aspect* of the ball–bearing which is an object aspect.

The notion of object aspects is thus intimately connected with the notion of parts. Therefore, it is only natural to defend it against the standard criticism directed towards extensional mereology.

8.3 Referring to non–actual objects

A reference to an object aspect is a reference to a part of an object existing in a possible world which is causally accessible from the actual world. The ontological commitment of object aspects is thus a commitment to objects and parts in potential worlds.

After presenting and motivating this view, I shall defend it against two competing views. The first is *Platonism* — the second is an empirical perspective which denies that we can predicate other objects than those being present.

I divide possible worlds into those which are *potential* and those which are not. A possible world is potential if it can be causally reached from the actual world. Worlds in the past and counterfactual worlds are non–potential. If a world is counterfactual, evolution has turned in such a way that this world cannot become the actual one. There may of course be potential worlds with much similarity to counterfactual ones. In 1922, divisional traffic manager Georg L. Eir predicted a Copenhagen Metro to exist in 1952. However, the Metro was not implemented until 2001. Thus, the possible world at year 1952 containing such a Metro were counterfactual to the world which did not; just as it is counterfactual to the actual world today, cause even though we have the Metro as an actualium today, other things are different. Thus, a building consisting of certain objects and with certain properties may exist in a potential world just as well as in a world counterfactual to that. In that case, the counterfactual condition makes no restrictions on the existence of that building but concerns the existence and identities of other objects. Lewis includes counterfactual worlds among the class of possible worlds without making this distinction like the one above. However, I believe the distinction is convenient in the present context.

The union set of the worlds described makes the set of possible worlds I commit myself to. However, this set is not as large as the set which Lewis seems to commit himself to [120]. Possible worlds in the wide Lewisean sense may include worlds inhabited by Hobbits and Wizards. Certainly, we could imagine that evolution had turned differently in ancient times such that creatures like these exist today. However, in order for such world to be believable, I think it must at least be possible to properly define the corresponding counterfactual conditions, based on the natural laws of our actual world. That is, I insist on natural laws to be necessary as well as consistent in all possible worlds.

Also — but then I might be too severe — I do not see how the existence of such worlds having other natural laws contributes to understanding the meaning of informal or formal descriptions of things to be.

8.3.1 Motivation

One of the reasons I introduce the notion of potential worlds is that many descriptions referring to object aspects are of *normative* kind. Normative descriptions like requirements and designs of buildings talk of something in a potential world — something which *can* be realised.

Consider the sentence "*the hole in the wall needs to be drilled with a 5 mm stone drill*". We may say that the hole stands in a part–whole relation to the wall. Obviously, that part (the hole) does not exist physically at the moment the sentence is uttered. In order to make sense of the sentence, I believe that at least two issues need clarification.

The first issue is that a hole is something which is not there. A clarification is here quite easy, though. The hole can be described mathematically, e.g. as a solid cylinder shape. Whether the filling is of solid material is not of importance as long as we maintain a proper definition of the concept hole. As argued by Jubien the distinction between what is substance and what is not may be merely by convention [114]. The space occupied by a piece of gold covers more space than the sum of the spaces occupied by each gold atom. In fact, any kind of material will be sparse in spatial occupancy.

The second issue is that — for convenience — part–whole relations may be taken to hold between concrete, actual objects which can be observed, compared, and empirically judged. In the present case, neither of the objects may exist physically, and certainly the hole does not at the moment the sentence is uttered. If part–whole relations concern spatial inclusion (and usually they do), referring to object aspects implies a commitment to possibilia.

Now, consider the sentence: "*all exterior walls of the building need to be primed*". Normative sentences like this one, are like assignments of values to variables in imperative programming languages. The meaning of the assignment sentence can be seen as consisting of two components. The first is the side effect which updates the variable environment with the variable name bound to the value. The second is the resulting value of the whole expression seen in its context. The resulting value of an assignment sentences may be defined as either having no type or having a Boolean type. That is, the truth–condition of the sentence has not genuine value.

A similar, perhaps naive analysis, claims that the corresponding fregean truth–condition of the normative natural language sentence will always give true. If we assume that the expression has fregean sense and allow it to have reference although that may not be an actual object, the condition for whether the sentence is true looses its power. We cannot falsify such a normative sentence using this approach — it is like a definition. Instead, we may consider the sentence to be true when it is the case that the walls *have* been primed. This is a kind of a *speech act* approach (described thoroughly by Austin and Searle [112, 150]) but in a setting more similar to Smith's [156].

To imagine a sentence like the above to be true as well as offering valuable information, requires an ability to deduce the causal connections leading to

worlds containing the reference. Therefore, we need to commit ourselves to such causal connections.

In general, a sentence which refers to object aspects makes sense if there is a potential world such that the corresponding speech act truth–condition is true. Note, however, that it is not the same as there simply being a potential world in which the sentence is true, as that sentence may be normative and needs to be analysed in a speech act way.

The rationality in this view can be seen if we read the sentence as a conjunction of "*there may be such a thing as a building*", "*this building contains things which are exterior walls*", and "*all these things are primed*". Each of these sentences make sense on their own, simply because it is possible to satisfy their truth–conditions. Formalising this as a statement over possible worlds gives:

$$\hat{\exists} w : \mathbf{W} \bullet w_0 \leq w \wedge (\exists x, y \bullet \mathit{building}(y) \wedge \mathit{ext\text{-}wall}(x) \wedge x \ll_w y \wedge \mathit{primed}(x))$$

The commitment to object aspects as possibilia calls for a wider definition of parts than previously considered. An object aspect can also be something like a surface of a concrete foundation. Such a surface exists spatially when that foundation is actual, as well as before and after its actuality. But the actuality of the foundation does not make its surface more actual. It is still an abstract notion just like the intersection point between two straight lines in two–dimensional geometry.

8.3.2 Platonism

A claim that there are such things as object aspects, just as there are imaginary and natural numbers, may be justified in a commitment to these as abstract objects. A prominent doctrine for justifying the existence of such objects is that of *Platonism*.

One of the most comprehensive, yet precise, definitions of Platonism, I have come across is the one given in Crispen Wright's paper "*Wittgenstein's Rule–following Considerations and the Central Project of Theoretical Linguistics*" [172]. It says, that Platonism is the view that the correctness of a judgement is independent of any opinion or conception of ours.

The observable — as through sense impressions — is thus only imperfect images, and the perfect objects (the platonic ideas like objects of *archetypes*) are to be found only outside the scope of space, time, and causal connections.

In mathematics, Paul Bernay's (according to [43]) characterises a platonist as:

"[one who] *postulates the existence of a world of ideas which contains all objects and relations of mathematics*".

However, neither definitions claim everything to exist in a platonic heaven. Neither do they define what exists and what does not in such a heaven. Basically, I believe that Platonism in these forms needs clarification on at least two issues.

The first is that if platonic ideas — being the objects we refer to — are abstract; how can we grasp them? To this objection, Balaguer has an answer which suggests a so–called *full–blooded Platonism*. Full–blooded Platonism is the view that for whatever we can cognise, there is a corresponding platonic idea [11]. But this is just then a doctrine which is equal to a possible worlds doctrine without causal connections and without counterfactuals.

The second objection is that if platonic ideas are abstract concerning space and cause, how can it be that we can describe the forms of such ideas or reason about whether their actual counterparts can be produced?

I believe that my possible worlds perspective with its focus on potentiality gives a better explanation. Primarily, the causal connections are important in the sense that these facilitate reasoning about object aspects. Just as our experience with the natural laws of physics guides us in hitting a ball with a baseball bat, so does causation (in a priori or a posteori) direct the cognition of mental images in designing and planning of objects like bridges and houses. Thus, object aspects may be taken to exist outside the scope of time but not outside the scope of space nor completely outside causal connections. I say *completely* as object aspects are not subject to physical change which is the usual understanding of being within the scope of causal connections.

8.3.3 Empiricism

I take it to be evident that some entities must be universal in order for humans to learn and make theories about the world. First of all, the natural laws must be something we can share knowledge about. If this is not the case, I believe that a common science is impossible. Rooted in the natural laws lies the problem of causation and in that the notion of necessary connections.

When it comes to object aspects, the question is what kind of entities are common in a similar sense and thus justify the existence of object aspects even though these cannot be observed and measured. Derived from this question comes that of what kind of knowledge we share when we agree on the existence of object aspects. A pure nominalistic perspective can thus be rejected on the

basis that the ontological realm we are considering is not rooted in language and thus cannot be reduced to language construct rearrangements. Following Lacey, there are two definitions of nominalism: (i) denying abstract entities, and (ii) denying the existence of universals (particularism) [116]. I take the above argumentation to apply to both.

It is, though, not unnatural to claim an empiricist perspective. Such a perspective claims that knowledge must come from experience. The question is now whether that experience relates to something causal that can be shared; i.e., whether we should accept the notion of necessary connection as universal.

There are basically two definitions of causation and these are both rooted in the notion of necessary connection [157]. The first is that of *constant conjunction*. The second is that of *counterfactuals*. The former says that a cause c is followed by an effect e, and that all such causes are followed by all such effects. This definition appears to be a problem for empiricists like Hume because there is nothing that proves that a necessary connection is not just an idea in our heads. The latter says that a cause c is followed by an effect e, and if c had not occurred, then e would not have either.

If there were no such thing as referring to potential objects, how could we make a distinction between what can be made and what cannot? Then such objects would just be ideas as would the causal connections leading to the idea being realised. The fact that we think we can reason about descriptions referring to object aspects would then be entirely imagination.

I believe that an argument for the existence of causal connections and thereby for object aspects, is the same as Russell's argument against nominalism, given in "*The world of universals*" and "*On our knowledge of universals*" [143, 142]. My customized argument goes as follows. When we hit a tree with an axe, we notice that it makes markings. However, if the axe is very blunt it may take a long time and a lot of strength to cut the tree. From experience, we know that we can speed up the process by making the axe sharp. However, we do not have to make that experience before each hit with the axe. As for Russell's universals, we see a patch which classifies an axe as sharp or blunt. From a soft touch of the edge of the axe, we may even be able to tell whether it is sharp enough to quickly cut the tree.

What I have spoken of are the causal connections between striking and cutting. If it is so that based on some experience I can just feel the edge in order to tell whether it is sharp enough — without cutting myself with the same strength as when I hit the tree the causal connection must be something more than ideas in my head. An empiricist view which denies causal connections as universal thus seems untenable.

8.4 The problem of arbitrary sums

The distinction between the two levels of mereology presents the problem of arbitrary mereological sums in two perspectives which differ significantly. If we remain on the logical level there are no restrictions on the kind of objects we can put together to form mereological sums. Algebraically, we can define the sum of any collection of objects, as only the names denoting the objects are of importance — not what such collections take or can be taken to denote. In other words, we are not obliged to justify such combinations.

However, it seems that including epistemological issues makes arbitrary sums a serious matter. We may need to commit ourselves to concepts of which the characterisation and determination in language gets messy or may not even be possible.

There is a general principle of compositionality in mereology which says that the mereological sum of two or more objects is itself an object. It may here be essential to conceptually distinguish the concept of the whole from the concept of the individuals constituting the whole. An element in a set (even though that element could be a set of something) cannot be of the same sort as the composition or set in which it is an element. I.e. a type T is distinct from the type $T \times S$ and $T-\mathbf{set}$, etc. This follows from Russell's paradox and elementary type theory.

The question is now whether we can have objects (being parts or sums) for which there is not a concept. To me that would be absurd: Every object must fall under a concept. However, this also means that every arbitrary mereological sum (being an object as well) must fall under a concept. The proposition implies that a commitment to the existence of arbitrary sums on the epistemological level is also a commitment to the existence of arbitrary concepts under which such sums fall. That is:

$$\textit{arbitrary-sums} \Rightarrow \textit{arbitrary-concepts}$$

This is quite a large commitment which — I believe — does not satisfy Occham's Razor. Having arbitrary concepts, I believe is a commitment which is difficult to live with for several reasons. To me, concepts are sparse, and thereby I agree with Armstrong and Williams who — according to Lewis — believe that properties and concepts are to be considered in a sparse theory [121]. Also, Russell considers his universals to be in a sparse realm. The problem of arbitrary concepts may appear if we try to characterise arbitrary concepts. It may be that language is simply not rich enough to make such characterisation precise enough for all such concepts.

Consider the sentence "*exterior walls and exterior surfaces of beams not placed at the south side, need to be painted twice*". The problem of finding a suitable concept under which the walls and surfaces referred to fall may be approached in two ways. One way is to suggest a concept which is general enough. However, such a general concept may certainly include too many objects. Another way is to assume the existence of a concept of which the corresponding truth–condition is a disjunction of two conditions: one for being an exterior wall and one for being a surface like the ones referred to. We may succeed with the latter way, but we are then back at the logical level as we have not added any information about the objects as a whole, which can be connected with reality.

Both ways suffer from the fact that they do not relate the sums to larger wholes. Even though we are able to characterise the objects in question (as a whole or not) we have failed to restrict the objects satisfying the truth–conditions, to only be parts of a certain whole; a certain potential building.

Even if we assume that we succeed in characterising and determining a concept under which the considered arbitrary mereological sum and only that falls, we have a problem. Namely to justify the existence of such a concept by stating why we then need it.

In my opinion, one of the rationalities in believing in concepts being abstract entities is that we gain a *one–over–many* principle. The *one–over–many* principle says that many objects may be alike in some way and that their alikeness is a universal entity.

In the given case, it seems like having the arbitrary concepts we are looking for, replaces the *one–over–many* principle with a *one–to–one* relation between arbitrary mereological sums and corresponding concepts. Then the idea of concepts is lost and we might be better off remaining on the logical level of mereology.

Thus, I reject a commitment to arbitrary mereological sums. However, I believe that for the considered sorts of sentences and expressions which refer to object aspects, we do not need a commitment to arbitrary sums. Thus, we can avoid the commitment to arbitrary concepts.

My solution is simple. For object aspects it is always possible to split up a sentence or expression such that object aspects of different kinds, are treated separately. A sentence like the above we can split into: "*exterior walls placed at the south side, need to be painted twice*" and "*beams not placed at the south side need, to be painted twice*".

As a general principle consider the following. Construct a number of definite descriptions; one for each exterior wall and ceiling object. Each definite description

characterises and designates a certain object, and including part–whole expressions restricts the interpretation space of such objects to the whole in question. Here, topology and geometry are essential notions. Each definite description should define the exact spatial extension of one object aspect, as well as putting it into context. Now, consider the conjunction of these definite descriptions. Just as each description denotes an object aspect, so does the composite description.

The principle works because the restriction of interpretation space is performed by part–whole relations to a larger whole which has to be assumed, and not by unary predication constituting to the definition of concepts. An object like my front door falls under the same concept door no matter what apartment it is a front door to. The restriction of it being a part of my apartment is stated by the fact that it stands in a part–whole relation to it.

Thus, I accept that arbitrary collections of objects can be seen as sums in language for convenient predication. However, they are not genuine mereological sums on the epistemological level even though we can analyse them on a logical level.

8.5 The problem of flux

In a sense, the problem of flux derives from the problem of arbitrary sums. As I argued in the previous section, a commitment to arbitrary sums implies a commitment to arbitrary concepts. We may insist that there is a concept under which an object falls in each stage of its development or change through its life–time. That could be one concept general enough for the object to fall under it even though it changes over time. However, if we wish to be able to distinguish the stages conceptually, we must commit ourselves to distinct concepts for each stage. This is a huge commitment — a commitment which clings to the idea of allowing arbitrary concepts. This is a real problem for sentences and expressions which intensively describe formation and elaboration of objects as in requirements and design documents. It turns out, though, that the more modest ontological commitment of object aspects avoids the most serious charges rooted in the flux problem.

The problem of flux is both an identity problem and a problem of ontological economy. The problem is rooted in the fact that objects change over time without necessarily changing identity. By change, I basically understand physical genuine change. Gaining or loosing parts or intrinsic properties of parts are to me sufficient conditions for genuine change. To a trope theorist like Simons,

these situations are of the same kind. However, I — who do not believe in tropes at a conceptual level — shall focus on the gaining and loosening of parts.

The classical example of the flux problem is that of *Theseus Ship*. Over time the various parts of a ship are exchanged with new ones: mast, planks, ceiling, etc. At a stage, all the original parts have been replaced, and the question is now whether the ship has maintained its identity. We may say that it has, but not necessarily. Consider now the situation in which a person has collected all the original parts of the ship and put them together to form another ship. This other ship may be said to be more original with respect to the parts but not the form, which has been maintained for the other ship.

Extensional mereology is subject to an objection saying that here we have a problem of ambiguity concerning the identity of Theseus Ship. Simons defends extensional mereology against this attack by replying that we should make a distinction between what is *matter–constant* and what is *form–constant*.

I shall defend the notion of object aspects against the attack but with another argument. First, I shall consider changes to objects as physical entities. Second, I shall consider the problem of conceptually capturing and distinguishing different stages of objects under development and formation. Third, I shall consider the above problem of identity and in this context a special issue of ambiguity concerning part–whole relations.

Consider the issue of physical change of objects. Since object aspects are abstract, I shall not consider situations in which physical objects undergo tiny changes over time. Time is not an issue for me here. This means that changes which occur like when I accidentally scratch the surface of a wall and get dust on my arm, can be avoided. It is a change of the wall in the sense that molecules are removed from the wall as a whole. However, object aspects necessarily need to abstract from such small causal changes due to the rejection of arbitrary mereological sums. The only causal connections which are necessary are those of which our knowledge makes the foundation for rational thinking and reasoning about objects.

Consider the issue of distinguishing objects in a development process. We may here face the problem that we cannot allow that an object's obvious change in development or formation is not reflected by well and precisely defined concepts for each of the object's stages. I shall here split up the discussion in three parts; each being a rejection of a commitment to such arbitrary concepts. As a result, the notion of object aspects is suggested to be distinguished from that of objects.

Consider the sentence: "*This bike is painted green*". One mereological understanding of what it means to paint a bike green is that it is the mereological

sum of two arguments: the bike and the green paint. Spatio–temporally this seems to be true; but it is not necessarily a convenient conception. The understanding of the sum as an operation which takes argument objects and gives a result makes us rashly commit to a new concept: that under which the painted bike and only that falls. The rôle of that concept is to define a clear distinction between the arguments — the bike and the paint — and the resulting green bike.

However, we cannot know how small the change from argument objects to resulting objects may be. Also, which one of the argument objects is conceptually closest to the result? That is, which argument is the object being painted? Or is it the paint which is being "biked"? We cannot tell as mereological sum is a commutative operation which treats its arguments equally. In many situations, the operation simply modifies one of the argument objects. In order to answer the questions, we thus need a means for measuring similarity conceptually. An approach for doing so is given by Gärdenfors in "*Conceptual spaces*" [86]. I shall not follow this approach as I do not believe that properties and concepts can be put into a metric space, epistemologically. It is, though, an interesting idea — not without applicability in a formal definition–oriented context.

If we insist on distinguishing sums from their parts, no matter how small the difference may be, we have quite an expensive commitment. It is not only expensive according to Occham's Razor. It is expensive due to the possibility that language may not be rich enough to make the distinction explicit. We might call for a simpler solution with a more modest commitment.

It is tempting to see a building as made up by a number of individual parts and understanding structure as appearing from something like mereological sum. The problem of identity, however, arises if we identify the same object with a component in a mereological sum as with the sum itself. Our conception of a wall may be nearly the same even though that wall has undergone changes like priming and painting. We may still refer to it as "*the wall*" in which case we would have something like $[\![\ \mathrm{a}\ ]\!] = [\![\ \mathrm{a{+}b}\ ]\!]$. But this is absurd if a and b are taken to be proper parts. My suggestion is that the wall as a bare object and the wall predicated with how many times it has been primed or painted, are simply different object abstractions. What remains — even though the object undergoes small changes — is our apprehension of it at a certain level of abstraction. This apprehension couples to it a general concept under which the object falls and a context to which it belongs. This context is a part–whole relation.

Finally, consider the issue of identity. The ambiguity of identity which arises for Theseus Ship is not an issue for object aspects. The reason is that we do not have the situation in which an object aspect is replaced with another and then

being part of another whole. That would undermine the idea of object aspects as being abstract. Certainly similar object aspects may be referred to in the description of distinct objects; even in descriptions of a certain object in different stages of development. However, we can never have the identity ambiguity as for Theseus Ship. The reason is that object aspects are not something an object gains or looses physically — only conceptually.

However, we may need to explain the fact that a part of a potential object can be considered to be part of several different wholes; depending on what perspective we apply.

Consider the relation in which hinges stand to a door. We might say that the hinges are part of the door frame in which the door is mounted. We can take off the door; then the hinges, the door frame, and the wall in which it is inserted, can be considered a whole. However, the door can also be considered a product in which case it may come with the hinges in a package. Thus, the hinges might as well be considered parts of the door. We say that the hinges overlap the wall as well as the door.

The case shows that we can have many different views on an object. Each view may consider it to stand in a part–whole relation to a certain whole. However, the descriptions which express such part–whole information should not be considered to be in conflict. As I see it, there is nothing conflicting in both seeing the hinges as part of the door frame and of the door. To me, part–whole relations are simply constituting to the identification of objects. Each of them is a view on the object, and a number of such views contribute knowledge for reasoning and identification. Thereby, an object aspect — with its relation to a certain whole — can be seen as a *mode of presentation*. That is, an object aspect is what is grasped when understanding a description concerning part–whole knowledge. Object aspects can thus be seen as mereological counterparts to fregean senses — what is expressed when referring to an object aspect are conditions for something mereological to hold for an object in a potential world.

8.6 Appendix: Other mereological issues

Besides the problem of arbitrary sums and the problem of flux, extensional mereology has faced other less serious charges. Two of these are the claim of non–transitivity and the claim of sum ambiguity. In the next sections, I shall give my perspectives on these issues. The section is, however, not really part of the defence of object aspects, but a general defence of mereology against these charges.

8.6.1 Non–transitivity

The question of whether part–whole relations are always transitive, I believe, is rooted in the question of how we understand functionality of objects. The understanding of a part as something being spatio–temporally included in something else, rules out odd cases of non–transitivity. In this respect, if a lock is not part of the house containing the door in which the lock is placed, something is wrong.

Objects are often seen as playing a certain rôle. The misunderstanding, I think (following Simons [155]), appears when confusing such rôles with the spatio–temporal distinction between parts and wholes. A rôle defines some restricted area of functionality. A connection between part–whole levels serves as some sort of path for what I would call *potential actions*. The connections are like lines of causal commands. A handle and a lock as a whole offers the functionality of maintaining the door closed even though it is exposed to wind pressure. Opening and closing the door are actions, but they are potential as they do not have to be performed. Pressing down the door handle which triggers the lock mechanism is a potential action all the same. If the handle is pressed down, the door can be opened without using destructive force. Thus, performing a potential action makes a set of other actions ready to be performed. This line of actions is different from the relation between the door and the house. That is, the rôle the door plays — the functionality it offers — is different and the effect of performing it is different as well. The door facilitates letting persons get in and out while still having the possibility to protect against cold winds and temperature drop. However, these lines of functionality between parts and wholes are not the part–whole relations themselves but causal connections between action and reaction.

8.6.2 Ambiguity of mereological sum

The problem of ambiguity of mereological sum arises when considering the same parts to constitute distinct wholes. Two versions may here exist. One in which we simply have an overlap like two sets having a non–empty intersection set, and one in which we have genuine incidence. It is the latter with which I am concerned. What is criticized is the extensional principle that wholes are equivalent if they have the same parts. An example given in [155] is that Family Robinson and Basketball Team Robinson are wholes which consist of exactly the same objects, namely the family members. However, the wholes are conceptually different; they play distinct rôles or have distinct functions. They may not just be two names for the same.

In the following, I shall assume that the collection of parts remains the same. The opposed case is treated under the problem of flux.

It seems reasonable that a collection of objects can be given a name by definition. Still, the name "*Basketball Team Robinson*" does not capture all truth of the fact that the family members make up a team with that name. We may have two categories of such a whole. The first is the whole which exist purely by virtue of having certain parts, and that the parts existing are sufficient for the whole to exist. The second is the whole for which we, in addition, require that the parts relate in a certain way.

The distinction between wholes in the two categories seems to be that wholes in the second also may contain possibilia; i.e. abstract objects. This seems strange as the second in some cases is defined by a stronger criterion of truth. However, consider the following. The Basketball Team Robinson is a certain constellation of the objects, which we name. But so is Family Robinson. The only difference is that we assumed the latter collection's actuality. Assume that Family Robinson exists and that we can point to the members. The Basketball Team Robinson then only exists as a possibilium. It can be formed for certain occasions but it does not have to.

Naming the collection of family members "*Basketball Team Robinson*" seems not to add further meaning of the collection besides a compositional meaning of words in the name. The same goes fore Family Robinson. In a sense, descriptions of object aspects are names for certain arbitrary spatial extensions. It is only the name which lays down the criteria for recognition.

Let us try to explore this further. Basketball Team Robinson is distinguished from Family Robinson by contextual activities: The members play basketball, e.g. at a tournament and are therefore referred to this way. But so is the Family Robinson because every member of Basketball Team Robinson is also a member of Family Robinson, and vice versa. The former seems to be a kind of subset — but not a subset of members. Perhaps a subset of properties or skills. However, each member of the family does not gain nor loose properties by being a member of Basketball Team Robinson, although training as such a team may increase the member's ability to coordinate their movement and team play. However, this would be equivalent to the increase of such ability for the members of Family Robinson. Rather, I think certain properties are emphasized when we grasp the meaning of "*Basketball Team Robinson*". That is, Basketball Team Robinson may work as a kind of filter through which we see Family Robinson and the family members. This filter may exclude certain properties and relations and thereby emphasize others: The member's ability to play basketball, their individual skills in this matter, their team work, etc.

Such a filter thus determines under what conditions something is included in something else. Mereologically, under what conditions an object is part of a whole. But this is not enough. What space extension of an object makes a part? Is there something more besides spatial inclusion and this filter of properties which justifies a part–whole relation? For the above case there is not, but what about others?

A justification of this kind is certainly not trivial as argued by Artale et al in [7]. Artale takes some kinds of parts to be quite similar to object aspects, and some of them are mereologically problematic. E.g. "the top surface of the car" and "*the right side of the table*" are descriptions denoting parts. The descriptions (as the ones denoting my object aspects) strongly need context in order to make sense. But notice that there is reference to some whole of which we consider a part.

Chapter 9

Properties and design

Abstract: Design is a notion in which language, metaphysics, and cognition merge. In order to establish a theory of design, three philosophical questions need answering. These questions concern what it means to describe, how we can refer to non–present objects, and on what basis we can predict the behaviour of the objects we describe. The questions are rooted deep in the philosophical issues of properties which are considered essential in any design process, and design representation. The three questions are treated by discussing a number of philosophical writings on concepts, language and meaning, the problem of sense without reference, and the notions of shared entities in context of communication and prediction. An ontological basis for relating designs and requirements is sketched, based on Shoemaker's ontology and epistemological approach to properties. Furthermore, the process of designing is related to the process of predicting and reasoning over objects existing in possible worlds which are causally reachable from the present world in which they are described. Discussions lead to a thesis which aims at breaking the classical word–world dichotomy. The thesis suggests that between descriptions in language, a denotational relation may hold and that objects (which are usually considered the denotations of descriptions), themselves may have denotations; namely the causal dispositions they possess.

9.1 Introduction

The theme with which I shall be concerned in this paper contains a number of the philosophical problems which underlie research and practice of design. I shall take the notion of design to be the intellectual and practical elaboration of representations of artefacts like houses and bridges.

The name "*The Eiffel Tower*" is usually taken to denote the Parisian tower constructed for the World Exhibition in 1889. The name is a representation of that tower, but the name alone may hardly have been sufficient for constructing the tower in the first place. Neither may it be sufficient for verifying its design according to requirements. Instead, a set of properties may have been stated, like "*a height of 115m*" and "*made of steel*". Properties like a–height–of–115m and concepts like tower are crucial in any requirements and design processes of artefacts like buildings.

An ontological treatment of a domain — like that of civil engineering and design — must address at least two questions: What entities are there, and which are the fundamental, atomic ones? Answering these questions defines a collection of categories of being — also known as an *ontology*. An ontology is the range for variables in a formalized mathematical discourse, and it is the collection of objects referred to by words and phrases in language [151] (s.46).

Ontological work is of interest when it comes to requirements and design in civil engineering. The reason is that we are here dealing with descriptions and communication of ideas, based on conceptions. Communicating knowledge of artefacts like the concept of a 12"–concrete–wall depends either on a common understanding of the concept or of the properties characterising it. Therefore, a thorough study of the domain of civil engineering and design should be supported by a study of the fundamental ontological entities which make descriptions of artefacts meaningful and useful.

9.1.1 Design related problems

The process of designing can be seen as a problem solving process in which a number of properties are ascribed to a potential object in order for that object to offer certain functionalities (see Chapter 5). Thus, the design process involves descriptions, communication, reflection, and reasoning over knowledge of objects which may not have physical presence. In all of these processes, the notion of properties seems to be elementary.

A study of the philosophical foundation for design must at least address the following questions: (i) What does it mean to describe something?, (ii) how can we describe something which is not present?, and (iii) on what basis can we predict behaviour and functionality of described objects? I shall consider these three questions as *design related problems*. They are problems rooted in philosophy of language and in metaphysics.

The first problem, I shall call *the problem of describing*. Treating this problem requires clarifications on three issues: The issue of concept and property, the issue of how meaning arises, and the distinction between abstract and concrete entities.

The second problem, I shall call *the problem of the absent artefact*; thereby following Galle's terminology [80]. The discussion is centered on how to refer to objects which are not present.

The third and last problem, I shall call *the problem of prediction*. It comprises several issues including that of objectivity, communication, resemblance, and the epistemology of properties.

9.1.2 Objects and properties

Properties are often considered essential ontological entities in characterisation, description, and distinction of objects in formal sciences as well as in daily life. The distinction between objects and their properties is a well known philosophical subject with roots in metaphysics and in philosophy of language.

In informal systems like natural language, properties play the rôle of representing and characterising things, ideas, and phenomena. This is done through what Church calls *naming* [46]. That is, properties are entities which are referred to by nouns or phrases in language which have certain meaning in characterizing the thing, idea, or phenomena in mind. Such characteristics are descriptions but not the characterized themselves. Therefore, the characterisation may cover not just one object but classes of objects as the descriptions necessarily are abstractions.

A formal representation of an artefact must get its meaning from somewhere. In a model–theoretical framework, we might assign meaning to artefact representations by paraphrasing these in terms of properties. In fact, I should here say 'representations of properties', as properties may be considered to be abstract. That is, properties may not be linguistic constructs, so we need to represent them nominally; e.g. as pairs of names (attributes) and value signs. Such signs may denote atomic entities which need to be taken as primitives and thus as

commonly understood concepts, and objects. These could be natural kinds like water and wood. The problem though is that formal systems cannot have any *a priori* knowledge of such notions; they need a nominal definition like representations of the extension of the concept or representations of characterising properties which then need to be assumed, or similar. The problem may start a regress here, and the question is thus to what extent we need to paraphrase.

In formal systems like software and hardware systems, representations of properties are essential for capturing the interaction between the tools, and the surrounding world being sensed, represented and simulated. The reason is that they are the only means for representing and processing real world knowledge. That is, we can only store and manage descriptions or models of world aspects. We cannot store and manage the world or world parts as physical objects in the causal realm.

9.1.3 Objectives

The present paper concerns ontological perspectives and commitments to properties, and their application as foundation for formal descriptions of artefacts as in requirements and design of buildings in civil engineering. By *formal* I here understand *non–ambiguous* with respect to semantics.

Most philosophical writings have the form of *perspective—defence*. However, for this paper I have chosen a different order as I see the various philosophical doctrines as each contributing to the discussion and understanding of properties in design. For each design related problem, I present and discuss a number of important philosophical doctrines and critiques which contribute to an understanding and clarification of that problem. However, the selection of doctrines is far from thorough and I have tried not to let this paper be just another general treatment of property theories; although it is difficult to avoid. But this also means that I had to be quite selective concerning the doctrines presented, and also that I cannot go into all details or include all critiques, replies to critiques, etc.

My own perspective is built up throughout the paper and approaching the three problems can be seen as three steps towards it. The discussions throughout the paper stepwise lead to my position that denoting phrases in language and the objects they denote may not be two separate categories. Objects may also be taken to denote something, so in the area of design of artefacts, the often assumed *word–world* distinction seems unsatisfactory.

9.2 The problem of describing

The problem of describing concerns how meaning arises from sentences, expressions, and descriptions which we take to denote design objects or their properties. Central to the problem is the notion of concept as in any area based on abstraction. I believe it is reasonable to require that there is something general to say about collections of objects, and that generality must be rooted in something.

The discussion of concepts (and thus also of properties) divides the problem of describing in two. The first is that of the connection between concepts and the linguistic structures of language, and how meaning arise as a connection. The second concerns whether concepts referred to in language can be abstract or need to be concrete. I shall call it *Benaceraff's Dilemma* after the similar problem of reaching the abstract entities of mathematics [151]. I shall exercise a similar problem for objects of design when treating the problem of the absent artefact.

9.2.1 Concept and meaning

I shall start with Frege's definition of concept, his distinction between concepts and objects, and his compositional approach to define meaning. Frege's perspective is subject to several critiques of which I shall present Kerry's, Ramsey's and Davidson's; followed by my own.

9.2.1.1 Frege's notion of concept

Frege takes an important step in analytical philosophy in "*Function and Concept*" [71]. The notion of analytical philosophy is the priority which puts language over thought. The only path to a philosophical account is thus through analysis of expressing such an account in linguistic constructs [58] (p.17).

According to Dummett, the paper marks a breakthrough by identifying truth–values as objects and by indicating a distinction between sense and reference — a distinction which was thoroughly examined in the succeeding work "*On Sense and Reference*" [58].

In "*Function and Concept*", Frege shows how logical analysis — like in mathematics — can be applied to natural language. He does so by first making

a distinction between functions and numbers. A function like $x^2 = 4$ is an equation which is satisfied by the numbers $\{-2, 2\}$. This set is considered the *extension* of the function. Usually, we would say that a distinction between functions and values lies in that functions contain placeholder names like x. However, to Frege this is an unsatisfactory definition as it confuses form with content — name with denotation. It may lead to the misunderstanding that $2^2 = 4$ is a function of 2 or of 4.

Replacing x with a value, establishes an instantiation relation between x and the value of which the sign replacing x denotes. However, as a general principle of syntactical substitution, this seems to start an infinite regress [51]. The reason is that only a name — not a value — can take the place of x. Frege tried to avoid the regress by saying that what surrounds x is a predicate and that predicates, as functions, are unsaturated (i.e. incomplete). They are so in the sense that they need instantiation of arguments in order to denote a single value. For functions, such a single value is a truth–value stating whether the argument satisfies the equation of the function. Contrary, numbers are self–contained objects and do not need such instantiation. The notion of unsaturation, Frege takes to apply both to functions and function–signs.

From mathematics, Frege moves to natural language in order to apply a similar logical analysis. The sentence "*London is the capital of England*" is a fact. However, it can be turned into a function by substituting "*London*" — denoting an object — with a logical name x. We have: "*x is the capital of England*". To Frege, such a sentence denotes a concept; namely the concept of being the capital of England. A concept is thus a special kind of function: One which yields a truth value.

Meaning in language now arises in a compositional way. Nouns like "*London*", "*8*" and "*Waverly*" denote objects, and verbs and adjectives denote or contribute to denoting concepts[1]. I shall call this Frege's "rule of distinction". Analysing the expression "*Scott is the author of Waverly*" recognises "*Scott*" and "*Waverly*" as words denoting objects, whereas "*author*" has a predicative nature and thus denotes a concept. Objects are said to *fall under* concepts. Thereby, we mean that applying the concept as a function to the objects yields the value true, meaning that they satisfy their truth functions; false otherwise. In the above case, Waverly, which is an object denoted by the name "*Waverly*", is part of a more precise concept; that of being the author of Waverly. Under this concept the denotation of "*Scott*" falls. If we represent that concept by the predicate *author-of-waverly*, we have the predicate logical expression *author-of-waverly*(*Scott*) which is true

[1] In general, Frege's notion of concept can be taken also to mean *property* as noted by Mellor and Oliver [55]. However, Frege also introduces the notion of *Characteristics* ("*Merkmale*") in "*On Concept and Object*", which are notions having predicative nature on certain concepts. A distinction may therefore be suggested. However, I shall not do so in this work.

if "*Scott*" really is the author of Waverly. Thus, the meaning of a phrase in natural language is a function of the meanings of its components.

Frege's work revolutionised the Aristotelean logic by introducing logical implication and quantification. Thereby, we are able to analyse general phrases like "*all mammals are vertebrates*". Such logical analysis and the calculi for expressing it is essential to description sciences which aim at characterising and reasoning over characteristics of objects. Computer science and design of artefacts are such sciences which are similar in this sense.

In design, logical descriptions and analysis, based on the notions of predication and concept, are crucial. They are so in the sense that design can be seen as a process in which ideas are sketched and communicated as representations of objects not yet physically existing. Self–communication involving reflection is here a special case. Furthermore, the notion of predication is important in any formal representation of design ideas and design reasoning. Following the Fregean notion of concepts, we may say that certain concepts are *design concepts*. A design concept could be something of which the extension contains the possible realisations of a design idea (simply understood as some mental image, abstract entity, or similar — in this case it does not matter which doctrine to apply).

However, Frege's sharp distinction between concepts and objects, and his compositional approach to semantics, are not without problems.

Besides what led to Russell's paradox and besides the problem of co–extensionality — which I shall not examine here — Frege's approach faces two other charges. The first — stated independently and differently by Kerry and Ramsey — is the objection that a sharp distinction between concepts and objects can be based on linguistics. The second — stated by Davidson — questions the informativeness of names in Frege's compositional approach to semantics.

9.2.1.2 Kerry's objection

Kerry is concerned with Frege's sharp distinction between concept words, and words denoting objects. The reason is that in language we often treat concept words as subjects in predication. An example is "*the concept horse*". The word "*horse*" denotes a concept under which all objects that are horses, fall. However, in the present case it is the subject, and thus it fails to fit Frege's distinction rule.

The problem is of ambiguity which seems to arise from the fact that a concept

cannot appear in linguistic form. The concept needs to be expressed in some syntactical form using predicates. According to Frege this is a limitation in language. In other words, concepts cannot be presented alone because of the predicative nature of names denoting them.

In "*On Concept and Object*", Frege defends his view although he admits a certain vagueness in his original presentation of the distinction [74]. His reply to Kerry is as follows. Let Φ denote a concept under which an object x falls. Furthermore, let us assume that x has (at least) the properties A, B and C. These three properties we could collectively denote by Φ. The three properties are *characteristics* (*"Merkmale"*) of the concept denoted by Φ. Thus, we have introduced two levels of predication where A, B and C are first–level concepts and Φ is a second–level concept [47]. However, also A, B and C may be characterised and thus subject to predication. Continuing in similar fashion may start an infinite regress — a regress Frege tried to stop by making the distinction between saturated and unsaturated entities [51].

9.2.1.3 Ramsey's trinity

In "*Universals*", Ramsey argues that a distinction between concepts and objects faces a more serious charge namely that such a distinction cannot be based on linguistic terms.

Ramsey's argument goes as follows. Consider a proposition aRb, where a denotes an object like Socrates and b denotes a universal like wisdom. R denotes the relation of a to b like in "*Socrates is wise*".

The proposition states a trinity: (i) that R holds between the terms a and b, (ii) that a has the property of standing in the relation R to b, and (iii) that b has the property that a has R to it. We might use the abbreviation $\phi x \equiv xRb$ making ϕ a predicate symbol meaning that the argument x has the property denoted by b.

The problem is now that if ϕ is a name for the property of x having relation R to b, then ϕx will be the assertion that x has this property. That is, it will be a predicate–subject proposition with subject x and predicate ϕ. This, however, is not identical to the relational proposition aRb; only one of the three ways of understanding the relation, namely (ii). We might as well treat b as the subject and a as the name having predicative nature. In (iii) b seems to be the subject — in (i) the relation is purely syntactical.

A predicate symbol ϕ can only stand alone when it corresponds to a real uni-

versal and not an abbreviation like ⟨has R to b⟩. This should be $\langle \phi x \rangle$ in order to distinguish it from the two–argument ⟨has R to b⟩; i.e. $R(x, y)$.

Thus, from a linguistic view point we cannot really make the sharp distinction between concepts and objects which Frege claims to be essential.

9.2.1.4 Davidson on informativeness

Frege's compositional approach to semantics is undermined by Davidson in "*Truth and meaning*" [51]. Davidson argues that a theory of meaning should take the form of a theory of truth. From Tarski, we know that if we have a sentence S which is placed in a context P, we can apply the filling: S ***is true iff*** P[2]. An example is "*an animal is a platypus if and only if it lays eggs and is a mammal*".

[2]In the paper "*The semantic conception of truth*", Tarski defines an infinite hierarchy of meta–languages where each language talks about another language on a lower level [160]. The claim is that the hierarchy is indeed infinite in height and we thus cannot reach a language for defining the meaning of everything.

Tarski's aim is to solve logical paradoxes in language. However, I shall not focus on that part of Tarski's contribution. Rather, I shall focus on the contribution of *Convention T* which shows that there cannot be an informative theory of truth.

The argument of Convention T goes as follows. Given a sentence in a language, there are many things we can say about that sentence. We can say that it is grammatically correct or incorrect, we can say that it consists of a certain number of words and letters, that it has no verb in it, and so on. For each such kind of judgement we can define a predicate which holds true in certain situations; false otherwise.

A special thing we can say about a sentence is that it is true. Actually, we might want to abstract from the notion of *truth* and simply replace "is true" with a predicate *T*. We can now try to define the predicate T. This is done by establishing a defining sentence of the form "⟨x is P⟩ is T if and only if x is P" (a T–sentence). Examples are: "⟨Snow is white⟩ is T if and only if snow is white", and "⟨Giraffes live on Mars⟩ is T if and only if Giraffes live on Mars". The principle is called *Convention T*. Armed with Convention T, we can pick up any theory of truth and test it. For example we can test whether King James Bible candidates to be such a theory of truth. Consider for example: "*Tony Blair is Prime Minister*". Since King James Bible nowhere mentions Tony Blair, it does not apply to the sentence just given. This means that the predicate T cannot be equivalent to "is found in the King James Bible". King James Bible thus fails the test of Convention T.

Tarski makes, however, an even stronger claim which is the one I am interested in here. Convention T is a so–called *correspondence theory of truth*. Consider again the form "⟨x is P⟩ is T if and only if x is P". The left–hand side talks about a sentence — the right–hand side about the world. However, this will not do as elimination of the syntactical quotation marks to the left of "*is T if and only if*", leaves us with two identical sentences. Convention T is the strongest test for judging whether a theory of truth holds. It is itself the closest we can get to a theory of truth by means of the T–sentences, but it fails to be informative as we simply have "*x is P*" on both sides. That is, it does not contribute with knowledge of what it means for a sentence to be true and neither what it means for x to be p. The conclusion is therefore that truth is indefinable.

But why not settle with S ***means that*** P then? Davidson's project is to approach a theory of meaning which establishes a *word–world* relation. Such a relation unavoidably involves the notion of truth. However, I shall concern myself only with Davidson's introductory observations which concern Frege's compositional approach to define meaning; not Davidson's own contribution.

Davidson's claim is that understanding the meaning of a sentence in terms of the meaning of its components may not always be informative. The problem appears if we wish to define the meaning of a sentence like "*Theaetetus flies*" in a compositional way. The meaning of "*Theaetetus*" could be represented by the name Theaetetus denoting an object. The meaning of "*flies*" could be represented by the predicate *flies*. We thus have something like the predicate logical expression *flies*(Theaetetus) as representing the meaning of the sentence. However, this formula does not contribute with any real semantic information as it is simply a syntactical rearrangement of the sentence in question. If we really want to capture the meaning of the expression we need to express or paraphrase what it means to be Theaetetus and what it means to be flying. Here, we may again face an infinite regress. However, I shall later argue that such paraphrases *can* contribute informatively, even though they cannot escape Tarski's result of truth being undefinable.

9.2.1.5 What does *x falls under c* mean?

Frege's understanding of concepts as truth functions is an important step in the development of formalisms for expressing designs. However, I believe an important point will be missed if we do not seek to elaborate further on the understanding. In a sense it may give an unsatisfactory account for the rôle played by concepts and properties in design of artefacts. The reason is that what we have with a fregean view is not completely what we want. Let me illustrate this by an example.

Consider a CD-player and a compact disc containing the first symphony of Beethoven. When playing the compact disc, we are able to verify that it is indeed this symphony which is on the disc. Playing the disc and hearing the music is (besides the subjective aspects) like a truth function. We can answer *yes* or *no* to the question: "*Does this compact disc contain the first symphony of Beethoven*?" The same goes for a Fregean understanding of concepts and properties. These are truth functions which answer questions like "*is this a table*" or "*is this chair made of wood*". The test of whether x falls under the concept c follows sequentially after interpretation of descriptions denoting x and c respectively.

However, in design, we need a means for going the opposite way. The goal is not the truth value of a sentence but to build that sentence. Following the example above, we need a means for producing the compact disc such that it *does* contain the first symphony of Beethoven.

Frege took his truth functions to be a kind of primitives in the sense that there was alot hidden in the functional arrow of the signatures representing the concept functions. We do not, with fregean concepts as functions, have explicit function definitions which state *how* to judge whether an object falls under a certain concept. If I write a computer program for sorting a list of natural numbers, the code expresses exactly *how* this is done. If we are going to express *how* to decide whether an object x falls under a concept c, additional ontological entities may be required. An account due to Shoemaker seems to provide an theory which is closer to what we want, and I shall therefore treat a part of Shoemaker's ontology in Section 9.4.3.

9.2.2 Benacerraf's dilemma

Just as numbers are abstract, so may properties and concepts be considered to be abstract entities. The problem is, however, to explain how we can have knowledge of such abstract entities which are outside time, space and causal nexus. The problem is known in philosophy of mathematics as *Benacerraf's dilemma* [151]. The dilemma is that of maintaining a realism to mathematical entities being abstract, and at the same time justify that mathematicians can have knowledge of what they talk about.

To me, Benacerraf's dilemma is just as important for objects in design as it is for objects in mathematics. The reason is that objects in design may not have physical presence, and thus we may need to accept reasoning over abstract objects as in mathematics. However, I believe that design is somehow linked to reality and thus we cannot accept a theory which reduces it to a nominalistic realm.

I shall start with Russell's commitment to universals — roughly his concepts — and his argumentation against nominalism. Then follows Quine's principle of paraphrasing as an escape from a commitment to abstract entities like Russell's universals. I shall include some of Quine's holism as well. Finally, I shall consider trope theory which I shall reject as an informative theory of properties in context of design. To me, there appears not to be any of the theories which really solves Benacerraf's dilemma — but each of the presented perspectives contributes various clarifications.

9.2.2.1 Russell on abstractness

Russell elaborates on Frege's notion of concepts and calls these *universals*. He does so both in "*The world of universals*" and in "*On our knowledge of universals*" [143, 142]. The notion of universals arises from the idea that many individual objects may fall into the same category of being. E.g. all yellow objects fall into a category of yellow and all objects tasting sour fall into a category of sour. The question of whether an object falls into a certain category is, Russell says, independent of human thinking or convention.

Russell thus attacks the nominalist rejection of concept and properties as being abstract entities. We may need to perceive an object in order to categorise it properly, and different people may have different conceptions or criteria for doing so. However, it is evident to Russell that the categories and the relation between objects and categories are universal. This means that an object possesses a property independently of any knowledge of the object or of the property. Similar remarks apply to relations like being south of Chicago. The notion of universals comprises concepts, properties and relations.

To Russell, a universal is abstract in the sense that it is outside the scopes of time, space and causal connections. E.g. the universal of human exists independently of our conceptions, knowledge of it, and of the objects falling under it. Also, it exists even though there might not be or might not have been humans.

Frege assumed his concepts to be abstract as well, so both Frege and Russell are realists in this sense. However, Frege did not put much focus on the distinction between abstract and concrete when defining the notion of concepts. But Russell did, as the *one–to–many* relation between universal and objects was essential for his argument against nominalism. If we wish to predicate an object as being yellow, that property is something which is not associated solely with one but with many objects at the same time.

Escaping from a commitment to a property denoted by a predicate like *white* requires instead a commitment to a *patch* for judging whether objects are white and for defining resemblance. Such a patch is a "*particular*" which we consider a kind of *archetype*; a typical example of something being white. But such a patch is itself universal then — so the attack on nominalism seems to stay strong in this sense. At this point it thus seems that the platonic principle of *one–over–many* remains.

Still, Russell does not explain *how* it can be that as sensing individuals we have the ability to reach abstract entities like universals. Thus, it seems like we are still trapped in Benacerraf's dilemma.

9.2.2.2 Quine's principle of paraphrasing

Quine denies the existence of concepts and properties as abstract entities in "*On What There Is*" [133]. He does not reject the notions of concepts and properties though, but believes that as abstract entities these harbour a problematic and unnecessary ontological commitment. An ontological commitment is the required existence of ontological entities necessary for a certain ontological perspective or representation to be consistent. Using the word "*Pegasus*" may commit us to a fact that Pegasus — the winged horse from the legend — exists. There are here two known solutions: (i) Pegasus just exists as a mental idea, or (ii) Pegasus exists as a possibilium (a platonic idea or an object in a possible world). Neither of these solutions are satisfactory to Quine. Using the word "*Pegasus*" does not have to be accompanied with a commitment to the existence of the creature. The reason — Quine says — is that we can avoid such a commitment by paraphrasing the name into a description. For "*Pegasus*" it could be "*whatever pegasizes*". Even though the example may seem silly, Quine has a point here.

What we often do in order to add meaning to words denoting concepts or properties, is to paraphrase what these concepts or properties are about. In many situations, we may succeed in replacing a name with a number of descriptions characterising the reference of the name. Thereby, even though not capturing all truth about an object, we capture several aspects which together may contribute to further knowledge and understanding. Thus, predicating an object as *red* does not necessarily commit us to the existence of *redness* — only to the object in question possessing that property.

Still, I believe that many objections have been and can be directed towards Quine's perspective. One of them is due to Jackson who believes that Quine's paraphrasing principle can lead to an infinite regress [105]. The words and expressions in a paraphrase may themselves refer to something which calls for further paraphrasing.

Jackson thinks that notions like concepts and properties should not be understood with language as the only source. E.g. the blue colour of the sky was present way before any descriptions or names in language existed. Also, nominalism faces some problems when it comes to resemblance. The sentence "*red resembles pink more than blue*" may commit us to the existence of these three colours. Paraphrasing it into "*anything red resembles anything pink more than anything blue*" yields trouble as a red ball is then taken to resemble a pink elephant more than a blue ball. Such inconsistencies may be possible to overcome if various incomparable properties are taken into account; here both colour and shape. We may then — again following Jackson — say "*colour–resembles*" in-

stead. However, for anything more than toy examples I believe it to be no easy task. Paraphrasing is thus a complicated but unavoidable task in descriptions of artefacts.

However, Quine's perspective cannot entirely be considered nominalistic even though he tries to escape a commitment to abstract notions like concepts by nominal paraphrasing. As an example, he allow sets which are abstract too. In "*Two Dogmas of Empiricism*", he argues for a more sophisticated perspective which lies between realism and anti–realism. His ontology, which he admits is not a deep one, puts existensional quantification in the middle. A term is said to denote something if and only if it can be replaced by a variable in existential generalisation. An object exists if and only if it is in the range of some variable. In fact, we may then say that everything can exist in which case Quine's perspective is like Platonism or possible worlds. However, he insists on a certain level of ontological economy, like reducing (if possible) everything to be sets. Thereby, he believes that phrases of a whole language can be paraphrased to refer only to sets [151].

In some sense, Quine's focus on existential quantification makes his perspective similar to my perspective of *object aspects*; see Chapter 8. However, whether a variable could be assigned was to me not the criterion for the existence of reference; rather it was the existence of causal connections leading to the existence of the reference.

9.2.2.3 Tropes

The introduction of trope theory can be seen as an objection to properties as abstract entities and seemingly as a solution to Benacerraf's dilemma for properties. If properties are objects for sensing, characterisation and comparison, then how can they be abstract and thus outside causal nexus as Frege and Russell claim? A Fregean—Russellian perspective is thus unsatisfactory for epistemological reasons. On the other hand, a nominalistic perspective seems to be too restrictive, and both doctrines may be accused for linking language with a mirage of reality. Trope theory attempts to solve the problem of reachability by suggesting that physical objects posses, not properties, but *instances of properties*. E.g. the *yellowness* of *this* flower and the *sweetness* of *this* candy. Such instances are called *tropes* by Williams, Campbell, Daly, et al [171, 41, 49].

What is sensed of an object is its tropes which are intimately connected to the object. Thus, tropes are what is measured in comparison, distinguishing, and judging similarities of objects. The tropes of an object is only connected to that object and remain even though similar tropes of other objects are lost. A trope

can belong to many different sets of tropes where a set of tropes corresponds to a property or concept.

There are basically two theories of tropes: the *substratum theory* and the *bundle theory*. The substratum theory says that objects consist of a so–called *substratum* to which tropes are attached. The substratum is a *bare particulum* which does not have any properties but exists on its own. The bundle theory rejects the existence of substrata and says that objects simply *are* collections (bundles) of tropes [49]. Both Williams and Campbell favour the former theory, while Daly prefers the second.

Williams suggests a the theory of tropes in "*On the Element of Being*" as he recognizes that the relation between parts and wholes as in mereology has similarities with the relation between characteristics and the objects they are characteristics of [171]. In his argumentation, he names the parts of a number of lollypop sticks; among these their flavours. Thereby, he applies a Simonsean part–whole perspective [155]. This perspective says that having the colour yellow is similar to have be composed of objects of which one is the *yellowness*, similar to a window being composed by pane and frame. Such yellowness can thus not be something shared in fitting a *one–over–many* principle. Just as the lollypops have distinct names — one unique name for each lollypop — so are their flavours unique instances of properties. If one lollypop is destroyed, the others still have their flavours. If a *one–over–many* principle is applied, it should apply as a relation between properties and tropes — not between properties and objects.

Tropes have two sorts of connection points: location and similarity. Any trope can belong to as many sets as there are combinations. Williams abandons the traditional set–inclusion understanding of concepts. He accepts the existence of properties and concepts, but these are not sets or classes of objects in possible worlds or platonic heavens. Nor are they truth–functions stating object containment. They are sets of tropes and the semantics of an object is simply its set of tropes; i.e. its roughness, whiteness, roundness, etc. The elements in the set are the objects for judgements of e.g. resemblance. In other words: In stead of putting Socrates into the class of humanity, we put the human trope in Socrates.

9.2.2.4 Trope theory and descriptions

Trope theory raises an important question about the reachability of properties. Thereby, the theory justifies itself on the empirical front, but I believe there are problems rooted in the theory when it comes to certain kinds of descrip-

tions. The question is whether trope theory really offers more than Frege's and Russell's theories of concepts and universals as abstract entities or Quine's perspective centred on variable assignment. My argumentation that it does not, goes as follows.

Let us assume the existence of something like tropes being somehow instantiated properties that we can sense. If we want to describe that a certain wall is grey, we could — following Williams — associate a name with it's greyness trope. However, because of the abstraction ingredient in any description like in requirements and design of artefacts, the name applied as predicate in such descriptions may range over many objects and not just one. E.g. the description "*my front door is brown*" applies to my current front door and the front door I need to get if the current one is destroyed.

Each object referred to in a design or requirements descriptions is a *potential object*. It is so in the sense that it can be taken to exist in a possible world which is causally reachable from the actual one (see Chapter 8). Such potential objects are taken to satisfy the described criteria, and the justification that they do so, I believe is rooted in causation rather than in empiricism. It is essential to formal and informal descriptions that what they represent is knowledge of objects to be and that such objects can be produced in many copies based on the same description. In other words: the description is an abstraction aiming at approximating a design idea nominally. However, then our name for the greyness trope simply *is* a predicate as it may apply to many distinct objects and not intimately be associated with a single object. If thereby, our identification of tropes by use of names is simply a way of predicating objects, we might as well commit ourselves to nominalism or realism with respect to properties.

I believe that what appears from the discussion is that when it comes to descriptions of potential objects like in requirements and design of artefacts like houses and bridges, trope theory just adds an extra ontological level, which does not do anything for us. Furthermore, I doubt whether all properties — represented by predicates — fit into a trope theoretical framework. As an example, take a red wall of 7 inches which is made of wood. Surely, we can talk of the *redness* of that wall. However, I believe it to be artificial to talk of the *wood*–ness of the wall. Talking of the *7-inches*–ness of a wall is necessarily an abstraction as we can never have exactly a width of 7–inches. Thus is cannot be a trope.

An ontological realm, like a doctrine of properties, should be judged on its applicability in the current sciences as well as on its ability to explain essential epistemological phenomena. Therefore, I agree with Daly that trope theory faces similar charges as Frege's theory of concepts and Russell's theory of universals [49].

9.3 The problem of the absent artefact

The problem of the absent artefact — as presented by Galle in "*Design as intentional action: a conceptual analysis*"— arises when trying to assign to a design description some sort of object which is to act as the reference of the description [80]. However, it is essential in the area of design that such references do not have actual presence and thus the question is whether design descriptions are meaningful.

In approaching the problem, my point of departure is Frege's important distinction between *sense* and *reference*. Frege, though, seems to abandon the idea that references to non–actual objects are meaningful. This leads to Russell's claim that they are, if we understand the descriptions differently. Lewis, taking a drastic position, also believes that references to non–actual objects are meaningful — we just need to quantify over possible worlds instead. Finally, I shall elaborate on the doctrine of possible worlds by making a distinction between potential and non–potential objects, as well as considering the notion of having meaning or making sense, to be a matter of whether the reference can be causally reached.

9.3.1 Frege's puzzle

In "*On sense and reference*", Frege takes the philosophy of language to a new level [72]. He does so by claiming a distinction between the *sense* and the *reference* of an expression. The sense of an expression is what is grasped when trying to understand the expression.

Frege is interested in how knowledge can arise from the relation of equality or identity. The relation in an equation like $a = b$ can easily be understood as a relation between the two symbols a and b, but that does not necessarily offer any valuable information. If $a = b$ states the fact that whatever a denotes is the same as whatever b denotes we may use the symbols interchangeably. That is, we could write $a = a$ and $b = b$. However, such expressions are logically tautologies and do not express valuable information, whereas $a = b$ is informative. The problem is also known as "Frege's Puzzle".

The meaning is not only defined by the relation between an expression and what that expression refers to. We need an intermediate ingredient in meaning which Frege calls the *sense* (*Sinn*).

An alternative way of presenting a Fregean argument for the introduction of

sense is by use of literary figures in disguise, like Dr. Jekyll vs. Mr. Hyde, Superman vs. Clark Kent, etc. I shall use Alexander Dumas' figure from "*The Count of Monte Cristo*, namely the young sailor *Edmond Dantes* who is cast into prison, but returns as the rich and powerful *Count of Monte Cristo* (*Monte Cristo* for short)to take revenge over those who falsely accused him. The Fregean argument for introducing sense as a solution to the puzzle goes as follows:

1. The sentence "*Monte Cristo is Monte Cristo*" is trivially true.
2. The sentence "*Monte Cristo is Edmond Dantes*" is, however, informative and surprising.
3. If (1) and (2) differ this way, although not on their truth value, they must possess different cognitive significance.
4. If the meaning of a word is directly the thing referred to, the two sentences cannot differ in cognitive significance.

The distinction between (1) and (2) comes from the fact that the enemies of Edmond Dantes in the beginning know and believe (1) but certainly not (2). When they are confronted with (2), it has a tremendous impact on them and their lives[3]. However, it is not because these persons are irrational that they believe (1) but not (2). It is simply because the two names "*Edmond Dantes*" and "*Monte Cristo*" represent different modes of presentation and the connection between them is not established.

The sense of "*Monte Cristo*" is the incredibly rich, mysterious person who seems to come from all around the world and who is admired by everyone. The sense of "*Edmond Dantes*" is the young sailor who is believed dead in prison.

Frege suggests that the notion of meaning is a two component entity consisting of *sense* and *reference*. Thereby, we have a trinity of relations: (i) between signs (phrases, words, symbols, etc.) and sense, (ii) between sense and reference, and (iii) between sign and reference. The third may, however, be considered artificial as we cannot reach the object being referred to but the sense of it.

Frege's theory is now based on the notion of sense. When trying to understand a phrase, the person grasps its sense. Sense is objective as is the thought to which it constitutes. Thereby, sense is to be distinguished from the *idea* which is the mental construct or "*image*" of the person and thus subjective. Thus, people

[3]Though it should be stated that it is in fact one of the conclusions in the book that these persons themselves have lead their life into misery; not solely because of the disclosure of (2). However, this is another matter.

may share the same senses and thoughts when trying to understand phrases, but they may have individual ideas[4].

According to Dummett, the notion of sense is intimately connected to the notion of truth [58] (p.15). When trying to understand a phrase, we grasp its sense which means that we try to apprehend what the reference is. There can here be many possible ways of apprehending and adding several non–conflicting senses of the same reference constitute to build up knowledge or believe of the reference, although it does not guarantee any full understanding.

Grasping a thought — as through grasping the sense of a phrase — is our way of apprehending the condition for something of the reference to be true. That is, sense is the ingredient in the component called meaning which is relevant for making truth–judgements over phrases in language [59] (p.233).

The notion of sense — in context of designing — can be seen as truth conditions for artefacts being described. The definite description "*the Copenhagen opera house at the harbour*" has senses which can be grasped. One could be the conditions for what an opera house is. These conditions may include the existence of certain walls interrelated so and so, acoustic features, or simply the possibility of opera events to take place in the building.

The question is now whether we should rush to the conclusion that design descriptions express Fregean senses. If we insist that such descriptions should be meaningful, I believe that Frege would say that we should not. He takes it to be a leak in language that we can have phrases with sense but not reference [72]. The thoughts constituted by such senses are — following Sainsbury — at best *mock thoughts* which are to be distinguished from real thoughts [144].

Still, I think that there is a certain fairness in having such a distinction. It seems fair that design descriptions do not have the same cognitive significance in value as descriptions which can be justified empirically. Often design ideas are unclear and vague which may be why certain iterations of describing and reflection are necessary.

Also, we should see Frege's denial in context of his quest when he wrote "*On Sense and Reference*". At that point, he was looking for ways to define the perfect logical language. In a perfect logical language, all proper names must refer to something in order to be object for justifications which are not merely nominalistic and syntactical. Such a language was to be a candidate for being

[4]Note, that Frege's notion of *idea* ("*Vorstellung*") which he — according to Kenny — takes to mean a person's mental image, should be distinguished from the notion of *platonic idea* [115]. In Platonism, ideas are abstract, idealised counterparts to objects in the actual causal realm.

part of a foundation for mathematics — a foundation Frege never managed to establish because of Russell's paradox [58].

9.3.2 Russell's definite descriptions

Russell's commitment to objects falling under universals (roughly his concepts) is essentially distinct from Frege's. In "*Descriptions*" and "*On Denoting*", he defines the notions of *definite* and *indefinite descriptions* in order to solve the referential problem [140, 141]. A definite description usually includes "*The*" as in "*The author of Waverly*" and thus denote objects, whereas an indefinite description like "beam of wood" is usually taken to denote a universal (i.e. a concept or property). In a sense, the distinction is similar to Frege's distinction between saturated and unsaturated, but I believe Russell's notions are restricted to a linguistic scope and are aimed to clarify logical analysis — not to serve as an ontology like Frege's.

The referential problem appears when we analyse definite descriptions. The sentence "*The planet between Mercury and the Sun*" is one which to Frege has sense but certainly not reference. Thereby, all definite descriptions in design have sense but not reference.

Frege's claim that such descriptions do not possess real cognitive value and may even be meaningless, is absurd to Russell. They are perfectly meaningful — they just have to be analysed differently.

Descriptions like "*The bridge connecting Denmark and Germany across Femarn Belt is 18 kilometres long*" is a definite descriptions with sense but lack of reference. At the time these lines are written the bridge is still only a political agenda.

A Russellian argument for why the description is still meaningful would go something like this. Splitting up the description gives the sentences (i) "*There is at least one bridge which connects Denmark and Germany across Femarn Belt*" and (ii) "*There is at most one bridge which connects Denmark and Germany across Femarn Belt*" and (iii) "*Whatever bridge connecting Denmark and Germany across Femarn Belt is 18 kilometres long*"[5]. The sentences (i) and (ii) are false but meaningful as we can assign truth values to them. The expression (iii) is likewise meaningful and furthermore it is true if we assume a certain margin of precision on the measurement of the minimum distance[6]. It seems to me that

[5]The measurement of 18 kilometres has been based on the estimate of $17.7km$ found in the Femarn Belt pre–investigation [159].

[6]We might have used the phrase "*[. . .] at least 18 kilometres long* as this is the minimum

Russell's does indeed clarify certain things concerning our analysis, but I believe we need more.

9.3.3 Quantifying over possible worlds

In "*On the Plurality of Worlds*", Lewis argues that from a metaphysical point of view there is no such problem as sense without reference. In his possible worlds doctrine, we can simply quantify over the sets of objects existing in possible worlds. Thereby, there will always be a possible world which contains the reference in question.

Lewis' thesis goes as follows[7]. The world we live in is just one of many possible ones. We can easily imagine that some state of affairs in this world could have been different; namely if certain things in history had turned out differently. Also, from the world we live in things happen due to causes and effects. Thereby, at least some worlds are causally connected.

Objects exist in various possible worlds but may have different properties in each. Thus Lewis' fictive monkey Brownie may be able to talk in one world — i.e. in one counterfactual world — but not in an other. A counterfactual world is one which is contrary to another (e.g. the actual one) due to the fact that certain events turned to a path leading to the counterfactual world, instead of to a path leading to the actual one. The conditions which determine the ways of turning are called *counterfactual conditionals*.

Following this doctrine, the world is constantly changing; in one sense into other worlds — in another into other states of affairs. In such other worlds things are different, have different properties, lions loose their appetite, people get older and babies are born.

When referring to objects, we simply refer to *possibilia* which are objects that exist in a possible world. For Lewis, there is not limit. If there is a world in which "*a exists*" is true, there is also a counterfactual world in which it is not. More controversial it becomes when Lewis claims the symmetry of this counter–world relation. If there is a world in which *a exists* is false, there is also a counterfactual world to that in which it is true. The various worlds containing

distance for a connection; a bridge across Femarn Belt could certainly be longer. However, it does not ruin the logical argument as such a description would be likewise true and thus meaningful. Thus, the Russellian argument stands.

[7]Lewis' theory of possible worlds and its foundation for semantics has much similarity with Kripke's. It is, however, by chance I came across Lewis' theory before Kripke's. Hence, it is Lewis' account which is treated in this paper. The context in which the notion of possible worlds is introduced in this paper, might as well consider Kripke's account, I believe.

the things inhabiting these worlds are thus to be understood as *modes*. Modality is a question of quantification for the following reason. When claiming the existence of blue swans, we quantify over the possible worlds for which there is one in which such swans exist [120] (p.5).

One of the standard critiques to possible worlds in Lewis' setting is that we may refer to anything then. There seems no ways of judging whether referring to a hospital with 10.000 storeys or a floating ceiling is rational or believable. This is a critique which is important to the problem of the absent artefact. However, Lewis (who does not focus on language like Frege and Davidson) does not pay much attention to this issue.

The commitment of Lewis', I believe, has similarities with a commitment to platonic ideas. The reference of a proper name may be taken to denote an object existing in a possible world or it may be taken to denote a platonic idea. A basic form of Platonism may here restrict to a platonic heaven only consisting of certain objects; not all objects we can name in language. However, in that case we again have the problem of sense without reference; namely if idealisation is needed in order for a representation to be connected to a platonic idea. That is, we still need an instantiation relation. Balaguer has a solution to this problem of *sparse platonic ideas*, which suggests what he calls *full–blooded Platonism*. By full–blooded Platonism, he understands Platonism such that whatever we can cognise, there is a corresponding platonic idea.

In my opinion Balaguer's full–blooded Platonism and Lewis' possible worlds are quite close to each other. They both permit reference to non–actualia by taking a relation between representations (mental or written) and the objects denoted, as a primitive.

However, there is a crucial difference between them which calls for favouring the latter. Platonism does not offer any ordering of worlds nor any causal structuring. This means that we have a problem explaining the rationality in committing to design objects if these are platonic ideas. In possible worlds, we could say that an artefact being designed exists and that the description of it is meaningful simply because there is a possible world succeeding to ours in which that artefact is present. In the following section, I shall elaborate on this idea.

9.3.4 Justification of *exists* based on causal nexus

What makes a design description of an artefact meaningful is not that its reference exists, but that its creation can be causally justified. This is the perspective I shall justify in the following.

Let us solely consider design descriptions which are meant to refer to artefacts. By an artefact, Hilpinen understands a physical object which is the product of human actions and which has been produced intensionally for some purpose [100]. I shall here make a (perhaps for a moment odd) distinction between *actual artefacts* and *non–actual artefacts*. As a subclass of the latter, I define the notion of *potential artefacts*.

Artefacts can be actual in which case we do not have any problems of lacking reference. The properties of such objects can be sensed and measured and thus be subject to reasoning and tests. If the artefact referred to is non–actual we could pose the existential question: Is there a series of causal connections which leads to a possible world in which the artefact is actual? If so, the artefact referred to is *potential*. My perspective follows a possible worlds doctrine which, however, is more restrictive than Lewis'. It is so in the sense that I shall not consider all the possible worlds Lewis does. That would include worlds in the past and counterfactual worlds — worlds in which objects hardly relate to design descriptions. I shall only consider worlds which are causally reachable from the actual world. These worlds, I name *potential*. This, however, does not mean that I totally reject the existence of other Lewisean worlds — just that they are not of interest to me in context of design.

My argument for introducing causal nexus on the agenda is as follows. Potential artefacts are the objects referred to in design descriptions. The description "*my wood–house placed at the lake*", I take to be meaningful because the creation of a house made of wood and placed so–and–so at some lake, can be justified causally. From experience, we know that houses can be made of wood and from observation and reasoning over various maps, we can show what placements the house in question can have in some future. Our knowledge and experience are, however, only vague conceptions of what is true.

It is our knowledge of causes and effects, based on properties and change, that guides our judgement. Still, these causes and effects I take to be universal and thus independent of human thinking. Thus, I believe in there being objective truth which relies on natural laws. E.g. it is a fact that in order for a beam to resist a certain load on the middle, its dimensions must be at least so–and–so.

However, not all design descriptions have meaning. Many of the trick–paintings of Escher depict so–called impossible building structures [102]. These paintings have Fregean sense but they do not have meaning as there cannot be a series of causal connections (physically, financially, etc.) which leads to the creation of such structures. And I am not talking about the possibility of these existing if the history or the natural laws had been different. Such speculations are hardly of any benefit to the design practitioner, besides as psychologically based inspiration, perhaps.

I take design descriptions to be linguistic approximations. It is only natural then to allow for certain assumptions in a justification of whether a design description has proper believable reference.

Schön argues in "*The Reflective Practitioner*" that in design processes, certain restrictions like being too close to a highway or fitting bearing elements into an apartment design, may be ignored such that the design process flows more freely [149]. Then, more sophisticated or even radical solutions may be reached.

However, when I speak of reference, I speak of the reference of a final design. But even so, there may be design descriptions which may be considered meaningless because of certain restrictions in the surroundings. However, note that I am talking about causal referents and in the worlds containing these, conditions may indeed be different and perhaps so that they accommodate earlier assumptions. E.g. we may assume a suitable building site, even though no such thing exists or seems likely to come into existence. That is, I believe we are allowed to ignore certain criteria when reasoning over the potentiality of artefacts.

9.4 The problem of prediction

Predicting the behaviour of an artefact can be seen as the act of applying a certain pattern. When we experience the flexibility of a number of beams we may deduce what properties are needed in order for a beam to have such flexibility. If such a pattern is not just a mental construct, there must be something which is shared and which explains that we can communicate and agree on this prediction. That is, there must be such notions as objectivity and shared entities in order to establish scientific theories about flexible beams — or other kinds of artefacts.

The notion of communication seems central to this discussion as the design practitioner works in a self–communication process and perhaps in collaboration with other practitioners. The designer is constantly reflecting over the design and the abilities of artefacts satisfying the design. Thus, communication and prediction are intimately related as predicting behaviour is a communication process with the design situation. Therefore, I agree with Galle as I agree with Schön, that communication is essential to the process of designing [149]. Prediction also enters into our discussion as the communication in design is directed. It is directed towards ascribing exactly the properties which will make the artefact behave in certain ways in certain situations.

However, in order for communication to be possible, something needs to be

shared. There must be ontological entities of which the knowledge guides our thinking, reasoning, analysis, and description. Thus, the problem of prediction is closely related to the subject of how we acquire knowledge and inductively apply it, and it is closely related to the epistemological issues of knowing what it means for an object to possess a property — a problem which briefly introduces the notion of causation into the discussion.

I shall approach the problem as follows. Frege's notion of sense can be seen as an argument for requiring something to be objective and shared in order for communication to be possible. From Frege and the notion of objectivity, I present Russell's argument for accepting universals in order to explain that we can have *a priori* knowledge. *A priori* knowledge is the kind of knowledge which has its justification independently of any experience of ours [116].

From Russell, I move to Shoemaker's perspective on properties which seeks to explain how we can know that an object has certain properties. I shall here focus on the part of Shoemaker's ontology which concerns what it means for an object to possess certain properties; a part which brings aspects of causation into the discussion.

From Shoemaker, I go further into the problem of how descriptions relate to what they describe. My intention is here to challenge the classical distinction between word and world as two separate categories of being.

A linguistic construct like a sentence or a description may have denotation but may itself also be a denoted entity. The same goes for physical objects like houses, walls, and doors. Such objects, which are often taken to be denoted entities of descriptions, may themselves be considered as such syntactical entities having denotations of their own. The notion of syntax and semantics thus seems to be a matter of mode.

9.4.1 Objectivity and shared entities

Communication is a central issue in design as design representations would be useless otherwise. During the design process, self–communication — as a special kind of communication — is performed. It involves analysis and reflection over descriptions intended to denote artefacts or aspects of artefacts. The question is, however, on what basis communication is done. I believe that stipulating communication as a central and necessary ingredient in design calls for values which are universally shared. By shared, I understand that they are within a common referential framework and not just private to one person. The knowledge of such entities are necessarily subjective, but I believe that some connection

to objectivity is needed in order to explain basic design issues. Such issues include meaningful prediction of the behaviour of artefacts by ascribing certain properties to them, and that communication included in the prediction process is possible at all.

Shared entities could be Frege's notions of concepts or sense, Russell's notion of universals, or notions not directly related to language. To Frege, sense is essential in explaining communication. Passing knowledge from generation to generation is one kind of communication process. The reason it can be done — Frege says — is that: "*Man–kind has a common store of thoughts*" [73].

Accepting Fregean sense to be included in this store or at least constituting to the elements — the thoughts — in the store, I take to imply that there is such a thing as universal truth. If two persons understand a phrase in similar ways, it is because they grasp similar senses. An equal account holds if information is communicated from one person to another. One person expresses an idea by means of symbols which have Fregean senses. The symbols are read and interpreted by the other person who grasps these. In that case, we shall say that the message has been delivered, but whether it has been understood correctly, we cannot say as we have taken the relation between thought and idea as primitive[8].

Here, Frege's anti–psychologism seems to pay off [58]. If we had followed psychologism, the judgement of whether a phrase in language had a certain meaning would be completely up to the interpreter. In that case, we would have difficulty explaining why interpretation is not just a random cognitive process, and the communication foundation would thus be on weak ground.

9.4.2 Russell's induction

In "*On our knowledge of universals*" and "*The world of universals*", Russell is interested in the learning part of properties and thereby he shifts to a more epistemological agenda [143, 142]. Through senses[9], we observe things and phenomena in the world. Of these things and phenomena we perceive various properties which we abstract to fall under certain universals. When observing additional things or phenomena having similar properties, we apply a principle of induction to recognize these as being similar. The means for performing the induction, Russell takes to be universal.

Russell applies his principle of induction in an explanation of why we can have *a*

[8] A model of communication in design due to Galle has been presented in "*Artefact Specification, Design, and Production as a Process of Communication*" [79].

[9] Not *Fregean* senses though.

priori knowledge. For example, how can it be that we see a resemblance between two pieces of white paper? This, Russell argues, is not because we have been told that they share the property of being white. That would yield explanation problems when regarding the first human being observing and reasoning about the world. The explanation is to be found in the existence of universals and the notion of *one–over–many*. The two pieces of paper are as concrete objects interpreted involving thoughts of which the semantics are non–mental ideas on an abstract level; the objects that fall under the concept white.

But to perceive the relation in the equality $2 + 2 = 4$ or that a specific house is white hardly explains our *a priori* knowledge of such propositions. We need an account of how we perceive and how the connection between the concrete (although perhaps potential) world and the abstract world of universals is established.

Thus, Russell's theory of properties as universals has some of the same problems as Frege's theory of them as functions.

9.4.3 Shoemaker's causation perspective

Shoemaker adds some new perspectives to the discussion on properties — perspectives which I believe clarify a number of issues in the area of design of artefacts.

In "*Causality and properties*", Shoemaker prompts the epistemological question of how we can know that an object has certain properties [153]. His answer also aims at explaining what it means for that object to possess a property — an issue which is important in the context of design, I belief. However, his treatment does not lead to a discussion of knowledge and believe but involves causation, although the notion of necessary connection is not in focus[10]. The main principle is that observing a property is to be affected by it.

Shoemaker concentrates on properties that are *genuine*. The broad notion of *Cambridge Properties* satisfies the criteria that loosing or gaining such a property results in a Cambridge Change[11] — simply that Fx at time t and $\neg Fx$ at time $t' > t$. This definition also applies to so–called *Mere–Cambridge* pro-

[10] A necessary connection — in causation — is a connection between causes and effects, claiming a genuine relation between causes and the effects they invoke. A first question in causation is whether there really is a connection which is necessary, or whether such connections are just imagined and ideas in our heads, based on our desire for a deterministic and reliable world.

[11] In "*The Cambridge Dictionary of Philosophy*" the notion of Cambridge Change is, however, defined more restrictively as a *non–genuine change* [8].

perties like me being such that a barn is burning 100 miles away. Thereby, *Mere–Cambridge* properties is a sub–class of the class of extrinsic properties. However, the topological relation between the barn and I, are hardly part of my identity. Thus, Shoemaker is only interested in properties of which gaining or loosing, results in genuine change of an object. That is, a change in the object's intrinsics and not for example a change of the objects position in space. Extrinsic properties, he excludes altogether and replaces them with relations.

The link between properties and the world is due to what Shoemaker calls causal powers. Thereby, properties can be thought of as functions from sets of properties to sets of powers. The definition is recursive but not cyclic as properties are not defined in terms of themselves. Causal powers can be thought of as functions from circumstances to causal effects.

For most such functions from state to state there is a range of values for which the state does not change. The important step is when the state *does* change. What makes the function from state to state informative is the fact that certain values mark limits at which e.g. a beam collapses because the load is too high. It is the investigation of such limits — lying at the heart of material and civil engineering sciences — which make our ontological perspective useful and applicable in systems aimed for design and modelling.

We thus have that properties can be considered to have denotations and likewise for causal powers. The approach — if put in a formal setting — has similarities with denotational semantics of programming languages.

A property (p:P) is thus constituted by powers (q:Q) defining the change in world state (s:S). Shoemaker's notion of circumstances could roughly be assumed to be of the same sort as the effect; i.e. (s:S). Formally, we could write the signatures of the semantic functions as:

$$\mathcal{M}_P : P \rightarrow (P\textbf{-set} \rightarrow Q\textbf{-set}) \tag{9.1}$$

$$\mathcal{M}_Q : Q \rightarrow (S \rightarrow S) \tag{9.2}$$

As an example, the meaning of being knife–shaped is that if the object possessing that property in addition has the property of being made of steel (for instance) the object has potential functionality to cut wood; cf. (9.1). And having the power to cut wood means that it is able to partition a block of wood if the object is stroked so–and–so towards the block; cf. (9.2).

We thus have a perspective in which properties are not the atomic entities to

which we commit ourselves. They are entities which only potentially can make a world change and they need circumstances in order to activate the causal powers.

What defines the identity of objects and thus the distinction is not the properties as shared entities but the causal powers as shared entities.

In this sense two individual objects have similar potential for functionality if they possess properties of which the causal powers denote similar changes. E.g. an instrument shaped like a star and an instrument shaped like an arrow may make similar markings in a plate of wood. This is not due to the shape properties being co–extensive but due to the fact that the causal powers they invoke are alike in denotation.

9.4.4 Elaborating on Shoemaker

In Shoemaker's ontology, I see at least three ideas which may clarify the notion of design and approach a solution to the problem of prediction.

First, properties and causal powers (which we shall call *causal dispositions*) seem to be the ontological entities referred to in design and requirement descriptions, respectively. The denotational relation between properties and causal powers thus seems to clarify the problem of design verification. By design verification — in context of artefact making — I understand the process of checking or proving whether a design meets a set of requirements.

Second, it may be that properties are of secondary importance in design cognition even though they are primary in design description. The denotational relation between properties and causal dispositions, I take to be the very essence of design rationality and the knowledge (even tacit) of it is what seems to guide us as rational practitioners. The dispositions are abstract in the sense that they are outside time, space, and causal connections, but they appear concretely in causal scopes as they are intimately connected with natural laws consistent in all possible worlds.

Third, the introduction of causal powers as something closely related to objects in the causal world, seems to break down the traditional dichotomy which classifies entities (objects or phrases in the realm of language) as either of denoting or denoted kind.

The meaning of an object can be considered to be a set of properties, and the meaning of these, a set of causal dispositions. Thus, indirectly, the meaning of

an object is its dispositions. Often objects are considered to be the references of descriptions. However, objects may themselves be denoting entities. Thus, the classical dichotomy dividing entities into either denoting kinds or denoted kinds, seems to be on weak grounds. The distinction between syntax and semantics may then be modes in which descriptions or objects are considered.

The idea seems to apply well to the context of artefact development. Here, we have descriptions at various stages. These descriptions may relate denotationally like requirements denote designs, designs denote potential artefacts, and artefacts denote causal dispositions.

I shall further discuss these elaborations on Shoemaker's ontology in the following sections. However, the ideas presented are tentative and are not supported by systematic argumentation as the discipline of philosophy requires. They are to be seen as propositions for future work.

9.4.5 Design verification

The problem of verifying designs comes out the problem that requirements (especially functional requirements) are expressed in terms which belong to a different set than the terms used for expressing designs. E.g. "*the wall must reduce noise by 25%*" is a functional requirement whereas "*the wall is to be made of concrete*" is usually considered a design choice. Only the latter sentence refers to the intrinsic properties of the wall considered. Certainly, we may have requirements which do refer to intrinsic properties like dimension properties and in these cases, the corresponding design statements delimits to single values in the property range, so I believe there is still a difference. In general, though, we cannot be sure that we can express the relation between design and requirements by set–inclusion of properties.

It seems though that Shoemaker's ontology, in which properties denote causal dispositions, opens up for a solution. The argument goes as follows.

A set of requirements to a building denotes the infinite set of possible designs. Each design satisfies the set of requirements by stating a composition of objects, properties of these objects, and relations between objects. Ideally, this is the case. However, we may not be able to express all design aspects in terms of formal, nominal, property designators. For now we shall assume that we can and that it is rational to do so.

The properties of the artefact being described have a meaning. Following Shoemaker, this meaning is the causal powers possessed by the artefact once pro-

duced. Also the causal powers denote something; namely functions from state to state which is the potential behaviour. Potential behaviour is in fact what we try to express in requirements — and especially in functional requirements. Thus, we can say that the causal powers of the artefact, *justifies* the requirements, and we have a structure as depicted in Figure 9.3. In the figure, the relation *refer–to* between design and properties means that designs are expressed in terms which refer to properties; it does not mean that designs denote properties.

$$\begin{array}{ccc} \textit{Requirements} & \xrightarrow{\textit{denote}} & \textit{Designs} \\ \uparrow \textit{justify / refer to} & & \downarrow \textit{refer to} \\ \textit{Causal powers} & \xleftarrow[\textit{denote}]{} & \textit{Properties} \end{array} \tag{9.3}$$

Properties of objects denote causal powers (dispositions) which are what is expressed in requirements. Expectations or aims at acquiring specific dispositions for objects is finding a certain constellation of properties such that it will be the case. Knowledge of the link between properties and dispositions is what construction and material science is all about, and these sciences aim at finding and describing such connections between properties and causal powers.

9.4.6 Prediction and causal nexus

The entities which are universal and thus explain communication, reasoning, and the rationality in prediction, may be causal dispositions rather than properties of things. Such dispositions are rooted in natural laws and I take them to be rigid in all possible worlds. If I design a beam for a load–bearing structure, I may ascribe it certain material and certain dimensions. These are properties intrinsic to the beam in mind. Thereby, I predict its behaviour and the reason it is rational is due to how elasticity and strength of such objects derives from the chosen material and dimensions. My knowledge thereof makes it possible for me to judge the rationality of my design.

In "*On the elements of being*", Williams claims that "Metaphysics is the thorough empirical science", and continues [171]:

> "Every item of experience must be evidence for or against any hypothesis of speculative cosmology, and every experienced object must be an exemplar and test case for the categories of analytic ontology."

In some sense I agree — here we have an example of metaphysics meeting empirical technical science. When I know $p(x)$, I can reason — following Shoemaker — about certain dispositions of x. What are universal and abstract are the dispositions; not necessarily the property p. Thus, it does not matter whether I put on a trope theoretical perspective or a Fregean–Russellian perspective.

Properties can be referred to by linguistic structures, but an important meaning of design descriptions are the dispositions they indirectly denote for an object. That is, the truth–conditions for whether artefacts satisfy the requirements. Thereby, prediction in design as well as in daily life, is justified by following the causal connections of artefacts put in certain situations. In one sense, we again have something like Russell's induction principle. Though it is not directed towards his universals which were properties, concepts, and relations. Rather it seems directed towards causal dispositions. In another sense, it seem like causal dispositions — in a denotational Shoemakerian setting — clarifies Schön's notion of *seeing as* [149]. In design and research, we may benefit from seeing certain dispositions of functionalities in objects rather than their properties. Schön exemplifies this by mentioning the case which showed that paint brushes of natural fibres were more functional than those made of artificial fibres. The reason for the discovery, Schön says, was that the brushes were *seen as* some kind of pump which lead to the distinction. Thus causal dispositions seem to be central to many sorts of reasoning in design as well as in science.

9.4.7 Escaping the word–world dichotomy

Consider a production drawing for a building. The drawing is an abstraction of the building it describes. The production drawing for The Øresund Bridge is a collection of symbols placed so–and–so and which means something. It has a meaning because workmen and engineers are able to construct the bridge, based on their interpretation of the drawing. Thereby, the drawing can be seen as a piece of syntax which has a denotation. Since the drawing on some points is open for interpretation and since we could build the bridge many times based on that same drawing, the drawing may denote an infinite collection of such potential bridges.

Once constructed, the bridge also has a meaning (and potentially also before its construction). It represents the function of effectively transporting people and go[o]ds across the belt. Observing this function over time gives a measure of the traffic over the bridge.

Let me try to go backwards too. What kind of entity denotes the production drawing? That may be the functional and space related requirements to the

bridge. Going further backwards, we may observe some needs or political decisions aiming at releasing traffic across the belt, strengthen cooperation, etc. Indirectly, it could be a desire for placing Denmark on the technological map.

It seems that considering an entity as syntactical, from a given object or linguistic form we can have one of many denotations. Consider for example Shapiro's case from "*Philosophy of Mathematics — structure and ontology* [151]:

> "Consider an imaginary economist who, while at work, speaks an impoverished version of (technical) English, a language that does not have the resources to distinguish between two people with the same income. Anything she says about a person P applies equally well to anyone else Q who has the same income as P. If she notes that P cannot afford the tuition at Harvard and that P is likely to get audited, then the same goes for Q. Someone who interprets the economist's language might apply the Leibniz principle and conclude that for those stuck with the impoverished resources, $P{=}Q$. That is, from the standpoint of the economist's scheme, people with the same income are identified and treated as a single object. To be fanciful, if the interpreter sees a certain woman, he might say (on behalf of the economist), '*There is the* $35.000'."

My point is that the ways of *seeing* something is some kind of filter which emphasizes certain properties or dispositions of concern in the current situation. There are many different ways of *seeing* — at least on for each class of entities being the denotations. Similar ideas are due to Galle and Bjørner [80, 27].

And we may go even further — claiming that physical objects can be syntactical entities too. The distinction between representation and thing being represented may be illusionary.

A design description is a representation of a potential artefact. However, many ways of describing exists, including formal and informal textual descriptions, scale models, drawings, and animations. Thus, a scale model of The Eiffel Tower is a representation of that tower and it may have been used during the design and construction of the tower. But how do we judge whether we have a representation of something or that something itself? I believe that we cannot. Thereby, the classification of entities being of denoting kind and entities being of denoted kind seems to be on weak ground. My argumentation is as follows.

The difference in size between the scale model and The Eiffel Tower of which it is a representation, is not the condition for that object to be the model and not what it models. Neither is it constituting a condition as there is no

such condition. The scale model might as well be a full–scale model, and its construction concerning parts and material may be such that it is identical to the *original* Eiffel Tower. The only way to decide that it is still just a model or copy is that it is placed on a different spot, but that is also illusionary. Over night, we might replace the real Eiffel Tower with its full–scale model without anybody knowing it. Let us say that nobody would be able to tell the difference — what is then the ontological distinction between representation and the entity being represented? None, I believe.

9.5 Closing

Lately, it has come to my attention that new trends in philosophy points at the possibility that denotation links span whole chain structures — an idea which fits well in my ontological perspective. However, I did not manage to find any suitable references yet. The traditional word–world dichotomy as an uncompromisingly and final (even universal) categorisation thus seems weak. I look forward to see further elaboration on this matter; especially for the following reason. The idea of denotation chains and the break–down of the traditional word–world categorisation, is rooted in a question scheme which is common to many different sciences: *what does it mean to have p(x)*? Sciences which already pose such questions are those of ontology, design of programming languages, knowledge representation, and especially domain engineering. A coupling between these and the area of civil engineering and design, I believe has a lot of potential — some which may reveal design support possibilities that we have not dreamt of. As a first step, the relation between requirements and design should be incorporated in support tools. As a second step, other sorts of knowledge should be included. Thereby, we are building up an ontology of civil engineering and design and this ontology is based on denotational relations between concepts — not on relations defined by convention.

Part V

Implementation

Chapter 10

A language–based design tool

In addition to the work presented in Chapter 7, a small prototype system for interpreting conceptual design models, has been developed. The system is a prototype in the sense that it only aims a demonstrating the principle of semantic parameterized interpretation. Design models are expressed in the language $\mathcal{L}_M$, and interpreted according to a semantics written in the language $\mathcal{L}_S$ and a calculus for a small subset of the language AutoLISP. We shall refer to this subset language as AutoLISP^{--}.

The implementation is a collection of parser and interpreter mechanisms which systematically process artefact models, semantics, and calculus semantics; and produces a view.

However, we have made the following limitations compared to the specification and presentation in Chapter 7.

- The calculus semantics only handles expressions in AutoLISP^{--}. However, extending this semantics or replacing it with another, makes it possible to produce many kinds of views.
- For conveniences with respect to the scanner of AutoLISP^{--} expressions, we have made a distinction between *unary* minus and *binary* subtraction

as used in $\mathcal{L}_M$ and $\mathcal{L}_S$. Still, the views which are produced are valid in conventional AutoLISP.

- The debug information is poor and simply raises exceptions if a parse error or lexical error occurs.
- We have not made any effort for optimisation what so ever. Rather, we have strived towards code which is close to the RSL specifications given in Chapter 7.

The view that can be produced are command expressions for displaying lines and rectangles in AutoCAD.

The implementation contains the following modules:

Common. Commonly used types and functions.

Lattice. Lattice operations.

MScan. Scanner for artefact models $m{:}\mathcal{L}_M$.

SScan. Scanner for semantics $s{:}\mathcal{L}_S$.

ALScan. Scanner for AutoLisp^{--} expressions $t'{:}\mathcal{L}_{AL}$.

Design. Parser for artefact models $m{:}\mathcal{L}_M$.

Sem. Parser for semantics $s{:}\mathcal{L}_S$.

AL. Parser for AutoLisp^{--}.

SPI. Interpreter mechanism.

These modules are ordered as shown on Figure 10.1.

The tool has been applied on a number of sample models and semantics. An example is the following model of an entrance section for a building:

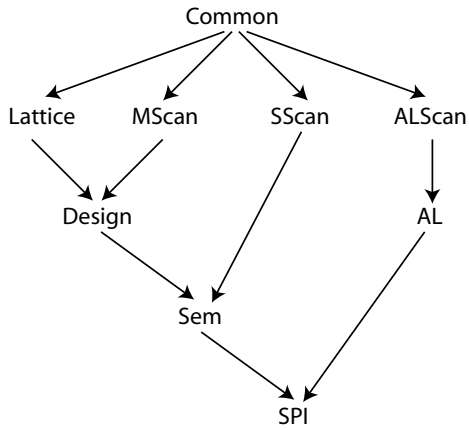


Figure 10.1: Hierarchy of modules.

```
model
  rsec :  (sort={rwall},
           width={450},
           height={340},
           thickness={70},
           ld={4},
           lw={10},
           ih={40},
           iv={30},
           relx={325});
  lsec :  (sort={lwall},
           width={450},
           height={340},
           thickness={70},
           ld={4},
           lw={10},
           ih={40},
           iv={30},
           relx={-325});
  csec :  (sort={obeam},
           length={280},
           height={30},
           thickness={70},
           ld={4},
           lw={10},
           rely={155})
```

where ld and lw are attributes for *lock depth* and *lock width* properties respectively, and ih and iv are the attributes for the horisontal and vertical measures for holes in the wall sections respectively.

The resulting view is a *wireframe* representation written in AutoLisp^{--}. Figure 10.2 and Figure 10.3 shows how AutoCAD displays two visualisations of this wireframe.

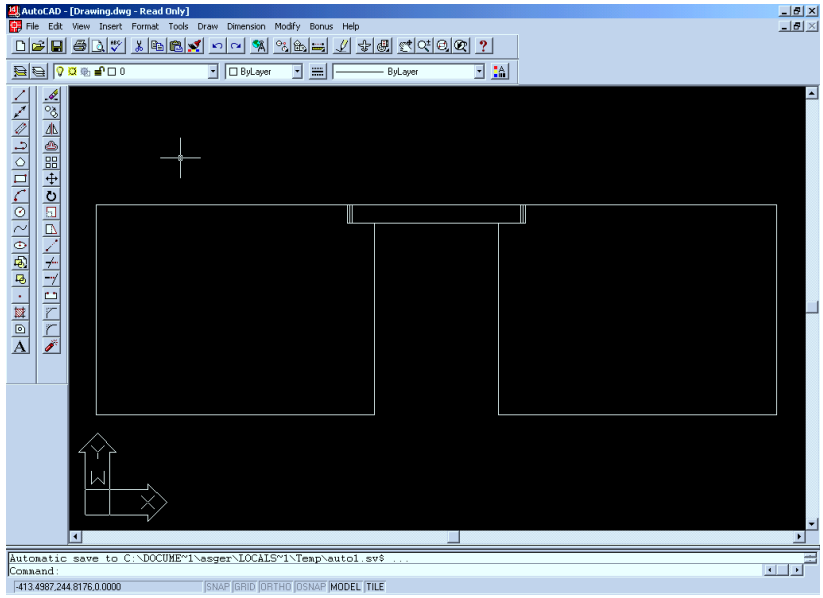


Figure 10.2: Front visualisation of entrance section.

Future work has been discussed in Section 7.10.1.

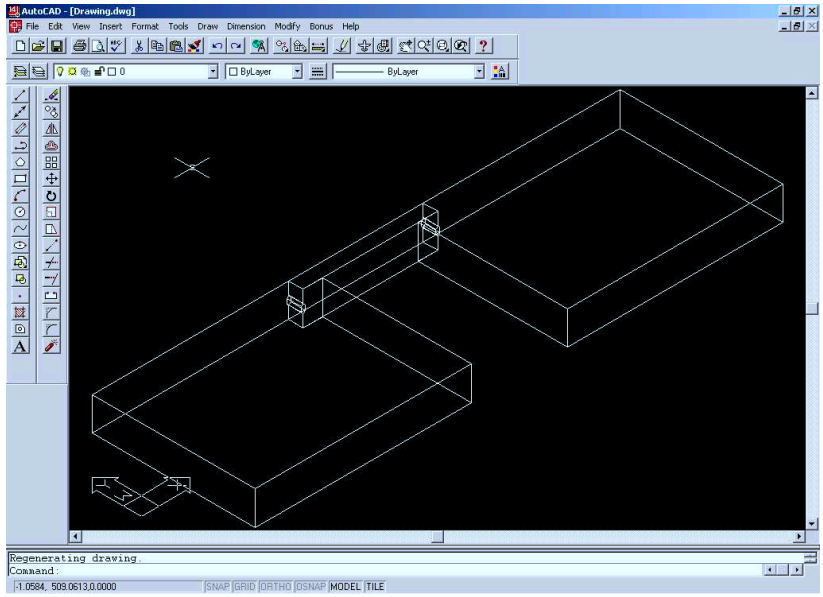


Figure 10.3: South–west isometric visualisation of entrance section.

Part VI

Closing

Chapter 11

Thesis results

It is time to compare the overall hypothesis from Section 1.2 with the overall results of the work presented. The individual chapters of the thesis make their own conclusions and considerations for future work. These conclusions and considerations are related to the subjects being treated in each chapter.

In this chapter, we shall consider the general conclusions of exercising the hypothesis from the four angles ①–④ defined in Section 1.2. In the following, we shall consider these angles in turn.

① **Relating concepts of different incomparable kinds.**

In Chapter 3, we have shown that the civil engineering concepts *cost frames* and *project plans* can be related as a Galois connection. This means that the concepts are not only related by their names but also by the mathematical structure which models the concepts. This makes the relation strong in the sense that its predicate can be given an explicit definition rooted in the intrinsics of the domain.

This Galois approach seems promising in two ways: (i) It establishes the formal foundation that civil engineering ontologies and concept systems need, and (ii) it makes the process of defining such ontologies and concept systems rigorous.

However, the approach is founded on the assumptions that:

- domain concepts can be represented as formal models and that these models capture the intrinsics of the real–world phenomena they abstract. The former may not always be the case, and the latter may be impossible to justify. Some domain concepts can only be modelled in a highly abstract way which may reduce the models applicability in a Galois connection.
- concepts in general relate to each other. However, there may be concepts which do not, and it is not certain what the criteria are for whether concepts do.

There is a great difference between stating a predicate for a relation, and defining how values of one concept can be calculated from values of another concept. The Galois approach does not deal with calculations in this sense. The reason is that in many cases such calculations are not possible. The approach is based on set–comprehension which are abstract specifications. Calculating a set of values — e.g. a set of project plans — requires that the abstract set–comprehensions are refined into algorithms for doing so. There may be cases in which such refinements cannot be performed or yields algorithms which are too complex to compute.

② **Relating representations of increasing cognitive significance.**

In Chapter 4, Chapter 5, and Chapter 6, we have shown that formal design models can be partially ordered in a lattice structure (a *design lattice*). Besides the mathematical implications this gives, the notion of *design lattices* opens up for new functionality in design tools. Such functionality includes browsing facilities and handling distributed designing in a structured way.

However, the notion of *design lattices* is founded on the following restrictions and assumptions:

- We have restricted design information to consist of references to a small set of basic ontological sorts: properties, values of properties, relations, values of relations, objects, and object decompositions. It may be that these ontological sorts are not significant, and that there are other concepts which are essential to design. These may include notions as knowledge, believes, and cross–discipline notions like Schön's *seeing–as* (see Section 2.3.1.1).
- Comparison of properties and relations respectively is based on set–inclusion of values. Holistic perspectives may claim the existence of more sophisticated notions. We have outlined this idea in other chapters by showing that incomparable concepts can be related by means of mediating ties. In Chapter 3, the ties were construction notions; in Chapter 9, they were metaphysical notions related to causation.

③ **Conceptual design models versus perspectives (*views*).**

In Chapter 4 and Chapter 7, we have shown that it is possible to calculate different views on conceptual design models. Such views can be representations of visualisations, expressions for stress calculations, etc. The relation between a conceptual model and the perspectives on that model is an example of a relation for which we can calculate values of the latter concept based on values of the former.

The study also clarifies the foundations for conceptual design tools that facilitate certain desired dynamic facilities. The principle of separating the semantics of design objects from the design program by turning the semantics into a parameter for the program, yields a design program which in a sense is generic. The semantics determines what can be designed and what perspectives can be put on a model. Thus, the architecture of semantic parameterised interpretation is a generic program. This program can be specialised by applying a semantics to it. Constructing or revising design programs is then a matter of writing or revising a semantics. Thereby, the principle of semantic parameterised interpretation "lifts" the notion of design tools one level in abstraction. This level is a *fix–point* in the sense that the generic design tool can be used to design anything — it all depends on the semantics we apply.

However, the approach of semantic parameterised interpretation is founded on the following restrictions and assumptions:

- The compositionality principle applied implies that there *are* perspectives which cannot be defined in this way (cf. Section 7.9 and 7.10.1).
- We also have the limitation that design models need to be on formal computable form. This may restrict a number of design ideas which seem only expressible in natural language.

④ **On the relation between descriptions and artefacts.**

We have — in two areas — outlined the philosophical problems and suggested some solutions, concerning the relation between language and artefacts. In general, we have shown how philosophical problems and approaches are relevant when studying the foundation for the domain of civil engineering and design.

In our study, we have considered the hypothesis in two contexts. One in which descriptions involve part–whole information (Chapter 8), and one in which we have been concerned with properties in context of design (Chapter 9).

In Chapter 8, we have suggested a metaphysical notion — *object aspects* — which we believe clarifies mereological issues in descriptions denoting artefacts.

In Chapter 9, we have shown how philosophical writings on properties and language contribute to a clarification on three design related problems.

Both papers lead to considerations of the relation between representations and the represented. In this context, we have shown in Chapter 9 that notions like requirements and design can be related by means of properties and causal dispositions. Thereby, the study of properties leads to similar conclusions as our study of how to relate civil engineering concepts by Galois connections.

We shall not try to falsify the philosophical perspectives presented. Philosophy moves in small steps and is a study of foundations. Philosophical perspectives are ideas of the world and not necessarily statements of which the truth value can be decided through empirical experiments. Different perspectives may explain different aspects of the world — also without being contradictious. The value of the perspectives presented may be measured on their theoretical or practical applicability as foundations for further ontological studies.

Chapter 12

Future work

We shall shortly outline suggestions for future work for the four different angles.

① **Relating concepts of different incomparable kinds.**

Future work includes:

- relating more civil engineering concepts and building up a formal ontology of civil engineering.
- putting such a formal ontology at work in computerized tools for document management, knowledge deductions, etc.
- investigating the ontological criteria which determine what domain concepts relate and which do not.
- exploring the mathematical and ontological properties of relations established in Galois connections.

② **Relating representations of increasing cognitive significance.**

Future work includes:

- applying the notion of *design lattices* in design tools.
- exploring the limitations of the presented languages, and the restrictions on ontological sorts which have been made.

- designing mathematical calculi for design representation and relating such calculi to existing object and function calculi.

③ **Conceptual design models versus perspectives (*views*).**

Future work includes:

- further exploration of formal design modelling languages and semantics.
- incorporating the principle of semantic parameterised interpretation in larger prototype design tools.
- gaining experience from real designing using such prototypes.
- generalising the interpretation mechanism such that any sort of target expression can be produced, and such that relations are taken into account.
- exploring the fix–point of design tool abstraction, and formulating the interpretation principle as mathematical equations.
- introducing notions from causation in interpretation of design models. Thereby, we combine results from Chapter 9 with ideas from Chapter 7. The idea is to consider cause–effects as special kinds of interpretations.

④ **On the relation between descriptions and artefacts.**

The philosophical studies in Chapter 8 and Chapter 9 can — as all philosophical studies — always be considered in more detail. Therefore, we shall not emphasize specific suggestions to future work for this angle. However, we believe that there are interesting issues to be discovered in the area of Shoemaker's causation approach to properties, and in the idea of breaking the word–world dichotomy as suggested in Section 9.4.7.

Appendix A

A short introduction to RSL

This is an ultra–short introduction to The RAISE Specification Language (RSL). The introduction is a slight modification of a "recap" from chapters in [30], and has kindly been lend out by its author, Prof. Dines Bjørner.

The "recap" is, alas, just an overview of the syntax of main aspects of The RAISE Specification Language. It intends to show some abstraction — that is model choices — possible in this language. For proper explanation of the semantics and pragmatics of the language, we refer to [134, 135, 30].

A.1 Type expressions

RSL has a number of *build–in* types (Boolean, integer, natural numbers, etc.) and type expressions (finite sets, infinite sets, Cartesian products, lists, maps, etc.). Let A, B and C be any type names or type expressions, then:

type

[1] **Bool**	[10] A^*
[2] **Int**	[11] A^ω
[3] **Nat**	[12] $A \xrightarrow[m]{} B$
[4] **Real**	[13] $A \to B$
[5] **Char**	[14] $A \xrightarrow{\sim} B$
[6] **Text**	[15] (A)
[7] A**-set**	[16] A | B | ... | C
[8] A**-infset**	[17] mk_id(sel_a:A,...,sel_b:B)
[9] A × B × ... × C	[18] sel_a:A ... sel_b:B

(save the [i] line numbers) are generic type expressions:

1. The Boolean type of truth values **false** and **true**.
2. The integer type on integers ..., -2, -1, 0, 1, 2, ...
3. The natural number type of positive integer values 0, 1, 2, ...
4. The real number type of real values, i.e., values whose numerals can be written as an integer, followed by a period (“.”), followed by a natural number (the fraction).
5. The character type of character values "a", "b", ...
6. The text type of character string values "aa", "aaa", ..., "abc", ...
7. The set type of finite set values, see below.
8. The set type of infinite set values.
9. The Cartesian type of Cartesian values, see below.
10. The list type of finite list values, see below.
11. The list type of infinite list values.
12. The map type of finite map values, see below.
13. The function type of total function values, see below.
14. The function type of partial function values.
15. In (A) A is constrained to be:

- either a Cartesian A × C × ... × D, in which case it is identical to type expression kind 9,
- or not to be the name of a built–in type (cf., 1–6) or of a type, in which case the parentheses serve as simple delimiters, eg: (A $\xrightarrow[m]{}$ B), or (A*)**-set**, or (A**-set**)list, or (A|B) $\xrightarrow[m]{}$ (C|D|(E $\xrightarrow[m]{}$F)), etc.

16. The (postulated disjoint) union of types A, B, ..., and C.
17. The record type of mk_id–named record values mk_id(av,...,bv), where av, ..., and bv, are values of respective types. The distinct identifiers sel_a, etc., designate selector functions.
18. The record type of unnamed record values (av,...,bv), where av, ..., and bv, are values of respective types. The distinct identifiers sel_a, etc., designate selector functions.

A.2 Type definitions

A.2.1 Concrete types

Types can be concrete in which case the structure of the type is specified by type expressions:

```
type
  A = Type_expr
```

The grammar for writing type expressions is:

```
[1] Type_name =
        Type_expr /∗ without |s or sub−types ∗/
[2] Type_name =
        Type_expr_1 | Type_expr_2 | ... | Type_expr_n
[3] Type_name ==
        mk_id_1(s_a1:Type_name_a1,...,s_ai:Type_name_ai) |
        ... |
```

```
          mk_id_n(s_z1:Type_name_z1,...,s_zk:Type_name_zk)
[4] Type_name :: sel_a:Type_name_a  ...  sel_z:Type_name_z
[5] Type_name = {| v:Type_name′ • 𝒫(v) |}
```

where a form of [2–3] is provided by combining the types:

```
Type_name = A | B | ... | Z
A == mk_id_1(s_a1:A_1,...,s_ai:A_i)
B == mk_id_2(s_b1:B_1,...,s_bj:B_j)
...
Z == mk_id_n(s_z1:Z_1,...,s_zk:Z_k)
```

A.2.2 Subtypes

In RSL, each type represents a set of values. Such a set can be delimited by means of predicates. The set of values b which has type B and which satisfy the predicate $\mathcal{P}$, constitute the sub–type A:

```
type
   A = {| b:B • 𝒫(b) |}
```

A.2.3 Sorts (abstract types)

Types can be sorts (abstract) in which case their structure is not specified:

```
type
   A, B, ..., C
```

A.3 The RSL predicate calculus

A.3.1 Propositional expressions

Let identifiers (or propositional expressions) a, b, ..., c designate Boolean values. Then:

false, **true**
a, b, ..., c
∼a, a∧b, a∨b, a⇒b, a=b, a≠b

are propositional expressions having Boolean values. ∼, ∧, ∨, ⇒, and = are Boolean connectives (i.e., operators). They are read: *not*, *and*, *or*, *if-then* (or *implies*), *equal* and *not-equal*.

A.3.2 Simple predicate expressions

Let identifiers (or propositional expressions) a, b, ..., c designate Boolean values, let x, y, ..., z (or term expressions) designate non–Boolean values, and let i, j, ..., k designate number values, then:

false, **true**
a, b, ..., c
∼a, a∧b, a∨b, a⇒b, a=b, a≠b
x=y, x≠y,
i<j, i≤j, i≥j, i>j, ...

are simple predicate expressions.

A.3.3 Quantified expressions

Let X, Y, ..., C be type names or type expressions, and let $\mathcal{P}(x)$, $\mathcal{Q}(y)$ and $\mathcal{R}(z)$ designate predicate expressions in which z, y, and z are free. Then:

$$\begin{array}{l}\forall\ \text{x:X} \bullet \mathcal{P}(x) \\ \exists\ \text{y:Y} \bullet \mathcal{Q}(y) \\ \exists\ !\ \text{z:Z} \bullet \mathcal{R}(z)\end{array}$$

are quantified expressions — also being predicate expressions. They are "read" as: For all x (values in type X) the predicate $\mathcal{P}(x)$ holds; there exists (at least) one y (value in type Y) such that the predicate $\mathcal{Q}(y)$ holds; and: there exists a unique z (value in type Z) such that the predicate $\mathcal{R}(z)$ holds.

A.4 Sets, Cartesians, lists, and maps

A.4.1 Set enumerations

Let the below *a*s denote values of type A, then the below designate simple set enumerations:

$$\begin{array}{l}\{\{\}, \{\text{a}\}, \{\text{a}_1,\text{a}_2,...,\text{a}_m\}, ...\} \in \text{A-}\textbf{set} \\ \{\{\}, \{\text{a}\}, \{\text{a}_1,\text{a}_2,...,\text{a}_m\}, ..., \{\text{a}_1,\text{a}_2,...\}\} \in \text{A-}\textbf{infset}\end{array}$$

The expression, last line below, to the right of the $\equiv$, expresses set comprehension. The expression "builts" the set of values satisfying the given predicate. It is highly abstract in the sense that it does not do so by following a concrete algorithm.

```
type
  A, B
  P = A → Bool
  Q = A ~→ B
value
  comprehend: A-infset × P × Q → B-infset
  comprehend(s,𝒫,𝒬) ≡ { 𝒬(a) | a:A • a ∈ s ∧ 𝒫(a) }
```

A.4.2 Cartesian enumerations

Let e range over values of Cartesian types involving A, B, ..., C (allowing indexing for solving ambiguity), then the below expressions are simple Cartesian enumerations:

```
type
  A, B, ..., C
  A × B × ... × C
value
  ... (e1,e2,...,en) ...
```

A.4.3 List enumerations

Let a range over values of type A (allowing indexing for solving ambiguity), then the below expressions are simple list enumerations:

```
{⟨⟩, ⟨a⟩, ..., ⟨a1,a2,...,am⟩, ...} ∈ A*
{⟨⟩, ⟨a⟩, ..., ⟨a1,a2,...,am⟩, ..., ⟨a1,a2,...,am,... ⟩, ...} ∈ Aω

⟨ ei .. ej ⟩
```

The last line above assumes e_i and e_j to be integer valued expressions. It then expresses the set of integers from the value of e_i to and including the value of e_j. If the latter is smaller than the former then the list is empty.

The last line below expresses list comprehension.

```
type
   A, B, P = A → Bool, Q = A ~→ B
value
   comprehend: A^ω × P × Q ~→ B^ω
   comprehend(lst,𝒫,𝒬) ≡
      ⟨ 𝒬(lst(i)) | i in ⟨1..len lst⟩ • 𝒫(lst(i)) ⟩
```

A.4.4 Map enumerations

Let a and b range over values of type A and B, respectively (allowing indexing for solving ambiguity); then the below expressions are simple map enumerations:

```
type
   A, B
   M = A m→ B
value
   a,a1,a2,...,a3:A, b,b1,b2,...,b3:B

   [ ], [ a↦b ], ..., [ a1↦b1,a2↦b2,...,a3↦b3 ] ∀ ∈ M
```

The last line below expresses map comprehension:

```
type
   A, B, C, D
   M = A m→ B
   F = A ~→ C
```

```
  G = B ~→ D
  P = A → Bool
value
  comprehend: M×F×G×P → (C ~m→ D)
  comprehend(m,F,G,P) ≡
    [ F(a) ↦ G(m(a)) | a:A • a ∈ dom m ∧ P(a) ]
```

A.4.5 Set Operations

value	examples
∈: A × A-**infset** → **Bool**	a ∈ {a,b,c}
∉: A × A-**infset** → **Bool**	a ∉ {}, a ∉ {b,c}
∪: A-**infset** × A-**infset** → A-**infset**	{a,b,c} ∪ {a,b,d,e} = {a,b,c,d,e}
∪: (A-**infset**)-**infset** → A-**infset**	∪{{a},{a,b},{a,d}} = {a,b,d}
∩: A-**infset** × A-**infset** → A-**infset**	{a,b,c} ∩ {c,d,e} = {c}
∩: (A-**infset**)-**infset** → A-**infset**	∩{{a},{a,b},{a,d}} = {a}
\: A-**infset** × A-**infset** → A-**infset**	{a,b,c} \ {c,d} = {a,b}
⊂: A-**infset** × A-**infset** → **Bool**	{a,b} ⊂ {a,b,c}
⊆: A-**infset** × A-**infset** → **Bool**	{a,b,c} ⊆ {a,b,c}
=: A-**infset** × A-**infset** → **Bool**	{a,b,c} = {a,b,c}
≠: A-**infset** × A-**infset** → **Bool**	{a,b,c} ≠ {a,b}
card: A-**infset** ~→ **Nat**	**card** {} = 0, **card** {a,b,c} = 3

- ∈ The membership operator expresses that an element is member of a set.
- ∉: The non-membership operator expresses that an element is not member of a set.
- ∪ The infix union operator. When applied to two sets, the operator gives the set whose members are in either or both of the two operand sets
- ∩ The infix intersection operator. When applied to two sets, the operator gives the set whose members are in both of the two operand sets.
- \ The set complement (or set subtraction) operator. When applied to two sets, the operator gives the set whose members are those of the left operand set which are not in the right operand set.
- ⊆ The proper subset operator expresses that all members of the left operand set are also in the right operand set.

- $\subset$ The proper subset operator expresses that all members of the left operand set are also in the right operand set, and that the two sets are not identical.
- $=$ The equal operator expresses that the two operand sets are identical.
- $\neq$ The non–equal operator expresses that the two operand sets are *not* identical.
- **card** The cardinality operator gives the number of elements in a (finite) set.

The operations can be defined as follows:

```
value
  s′ ∪ s″ ≡ { a | a:A • a ∈ s′ ∨ a ∈ s″ }
  s′ ∩ s″ ≡ { a | a:A • a ∈ s′ ∧ a ∈ s″ }
  s′ \ s″ ≡ { a | a:A • a ∈ s′ ∧ a ∉ s″ }
  s′ ⊆ s″ ≡ ∀ a:A • a ∈ s′ ⇒ a ∈ s″
  s′ ⊂ s″ ≡ s′ ⊆ s″ ∧ ∃ a:A • a ∈ s″ ∧ a ∉ s′
  s′ = s″ ≡ ∀ a:A • a ∈ s′ ≡ a ∈ s″ ≡ s⊆s′ ∧ s′⊆s
  s′ ≠ s″ ≡ s′ ∩ s″ ≠ {}
  card s ≡
    if s = {} then 0 else
    let a:A • a ∈ s in 1 + card (s \ {a}) end end
    pre s /∗ is a finite set ∗/
  card s ≡ chaos /∗ tests for infinity of s ∗/
```

A.4.6 Cartesian operations

```
type
  A, B, C
  g0: G0 = A × B × C
  g1: G1 = ( A × B × C )
  g2: G2 = ( A × B ) × C
  g3: G3 = A × ( B × C )

value
  va:A, vb:B, vc:C, vd:D

  (va,vb,vc):G0,
  (va,vb,vc):G1
  ((va,vb),vc):G2
  (va3,(vb3,vc3)):G3

decompostion expressions
  let (a1,b1,c1) = g0,
      (a1′,b1′,c1′) = g1 in .. end

  let ((a2,b2),c2) = g2 in .. end
  let (a3,(b3,c3)) = g3 in .. end
```

A.4.7 List operations

```
value
  hd: A^ω ~→ A
  tl: A^ω ~→ A^ω
  len: A^ω ~→ Nat
  inds: A^ω → Nat-infset
  elems: A^ω → A-infset
  .(.): A^ω × Nat ~→ A
  ⌢: A* × A^ω → A^ω
  =: A^ω × A^ω → Bool
  ≠: A^ω × A^ω → Bool

hd⟨a1,a2,...,am⟩=a1
tl⟨a1,a2,...,am⟩=⟨a2,...,am⟩
len⟨a1,a2,...,am⟩=m
inds⟨a1,a2,...,am⟩={1,2,...,m}
elems⟨a1,a2,...,am⟩={a1,a2,...,am}
⟨a1,a2,...,am⟩(i)=ai
⟨a,b,c⟩⌢⟨a,b,d⟩ = ⟨a,b,c,a,b,d⟩
⟨a,b,c⟩=⟨a,b,c⟩
⟨a,b,c⟩ ≠ ⟨a,b,d⟩
```

- **hd** Head gives the first element in a non–empty list.
- **tl** Tail gives the remaining list of a non–empty list when Head is removed.
- **len** Length gives the number of elements in a finite list.
- **inds** Indices gives the set of indices from 1 to the length of a non–empty list. For empty lists, this set is the empty set as well.
- **elems** Elements gives the possibly infinite set of all distinct elements in a list.

- $\ell(i)$ Indexing with a natural number, i larger than 0, into a list ℓ having a number of elements larger than or equal to i, gives the i'th element of the list.
- $\frown$ Concatenates two operand lists into one. The elements of the left operand list are followed by the elements of the right. The order with respect to each list is maintained.
- = The equal operator expresses that the two operand lists are identical.
- $\neq$ The non–equal operator expresses that the two operand lists are *not* identical.

The operations can also be defined as follows:

```
value
  is_finite_list: A^ω → Bool

  len q ≡
    case is_finite_list(q) of
      true → if q = ⟨⟩ then 0 else 1 + len tl q end,
      false → chaos end

  inds q ≡
    case is_finite_list(q) of
      true → { i | i:Nat • 1 ≤ i ≤ len q },
      false → { i | i:Nat • i≠0 } end

  elems q ≡ { q(i) | i:Nat • i ∈ inds q }

  q(i) ≡
    if i=1
      then if q≠⟨⟩ then let a:A,q′:Q • q=⟨a⟩⌢q′ in a end else chaos end
      else q(i−1) end

  fq ⌢ iq ≡
      ⟨ if 1 ≤ i ≤ len fq then fq(i) else iq(i − len fq) end
      | i:Nat • if len iq≠chaos then i ≤ len fq+len end ⟩
    pre is_finite_list(fq)

  iq′ = iq″ ≡ inds iq′ = inds iq″ ∧ ∀ i:Nat • i ∈ inds iq′ ⇒ iq′(i) = iq″(i)
```

$$\mathrm{iq}' \neq \mathrm{iq}'' \equiv \sim(\mathrm{iq}' = \mathrm{iq}'')$$

A.4.8 Map operations

```
value
  • ( • ): M → A ~→ B, m(ai) = bi
  dom: M → A-infset [ domain of map ]
      dom [ a1↦b1,a2↦b2,...,an↦bn ] = {a1,a2,...,an}
  rng: M → B-infset [ range of map ]
      rng [ a1↦b1,a2↦b2,...,an↦bn ] = {b1,b2,...,bn}
  †: M × M → M [ override extension ]
      [ a↦b,a′↦b′,a″↦b″ ] † [ a′↦b″,a″↦b′ ] = [ a↦b,a′↦b″,a″↦b′ ]
  ∪: M × M → M [ merge ∪ ]
      [ a↦b,a′↦b′,a″↦b″ ] ∪ [ a‴↦b‴ ] = [ a↦b,a′↦b′,a″↦b″,a‴↦b‴ ]
  \: M × A-infset → M [ restriction by ]
      [ a↦b,a′↦b′,a″↦b″ ]\{a} = [ a′↦b′,a″↦b″ ]
  /: M × A-infset → M [ restriction to ]
      [ a↦b,a′↦b′,a″↦b″ ]/{a′,a″} = [ a′↦b′,a″↦b″ ]
  =,≠: M × M → Bool
  °: (A ~m~> B) × (B ~m~> C) → (A ~m~> C) [ composition ]
      [ a↦b,a′↦b′ ] ° [ b↦c,b′↦c′,b″↦c″ ] = [ a↦c,a′↦c′ ]
```

- $m(a)$ Application gives the element of which a maps to in the map m
- **dom** Domain/Definition Set gives the set of values which *maps to* in a map.
- **rng**: Range/Image Set gives the set of values which *are mapped to* in a map.
- † Override/Extend. When applied to two operand maps, it gives the map which is like an override of the left operand map by all or some "pairings" of the right operand map,
- ∪ Merge. When applied to two operand maps, it gives it gives a merge of these maps.
- \: Restriction. When applied to two operand maps, it gives the map which is a restriction of the left operand map to the elements that are not in the right operand set

- / Restriction. When applied to two operand maps, it gives the map which is a restriction of the left operand map to the elements of the right operand set.
- = The equal operator expresses that the two operand maps are identical.
- ≠ The non–equal operator expresses that the two operand maps are *not* identical.
- ° Composition. When applied to two operand maps, it gives the map from definition set elements of the left operand map, m_1, to the range elements of the right operand map, m_2, such that if a, in the definition set of m_1 and maps into b, and if b is in the definition set of m_2 and maps into c, then a, in the composition, maps into c.

The map operations can also be defined as follows:

```
value
  rng m ≡ { m(a) | a:A • a ∈ dom m }

  m1 † m2 ≡
    [ a↦b | a:A,b:B •
      a ∈ dom m1 \ dom m2 ∧ b=m1(a) ∨ a ∈ dom m2 ∧ b=m2(a) ]

  m1 ∪ m2 ≡ [ a↦b | a:A,b:B •
              a ∈ dom m1 ∧ b=m1(a) ∨ a ∈ dom m2 ∧ b=m2(a) ]

  m \ s ≡ [ a↦m(a) | a:A • a ∈ dom m \ s ]
  m / s ≡ [ a↦m(a) | a:A • a ∈ dom m ∩ s ]

  m1 = m2 ≡
    dom m1 = dom m2 ∧ ∀ a:A • a ∈ dom m1 ⇒ m1(a) = m2(a)
  m1 ≠ m2 ≡ ∼(m1 = m2)

  m°n ≡
    [ a↦c | a:A,c:C • a ∈ dom m ∧ c = n(m(a)) ]
    pre rng m ⊆ dom n
```

A.5 λ–calculus and functions

RSL support function expressions for λ–abstraction.

A.5.1 The λ–calculus syntax

```
type /∗ A BNF Syntax: ∗/                value /∗ Examples ∗/
    ⟨L⟩ ::= ⟨V⟩ | ⟨F⟩ | ⟨A⟩ | ( ⟨A⟩ )      ⟨L⟩: e, f, a, ...
    ⟨V⟩ ::= /∗ variables, i.e. identifiers ∗/⟨V⟩: x, ...
    ⟨F⟩ ::= λ⟨V⟩ • ⟨L⟩                      ⟨F⟩: λ x • e, ...
    ⟨A⟩ ::= ( ⟨L⟩⟨L⟩ )                      ⟨A⟩: f a, (f a), f(a), (f)(a), ...
```

A.5.2 Free and bound variables

Let x, y be variable names and e, f be λ-expressions.

- ⟨V⟩: Variable x is free in x
- ⟨F⟩: x is free in $\lambda y \bullet e$ if $x \neq y$ and x is free in e.
- ⟨A⟩: x is free in $f(e)$ if it is free in either f or e (i.e., also in both).

A.5.3 Substitution

In RSL, the following rules for substitution apply:

- **subst**([N/x]x) ≡ N
- **subst**([N/x]a) ≡ a
  for all variables a≠x.
- **subst**([N/x](P Q)) ≡ (**subst**([N/x]P) **subst**([N/x]Q)).
- **subst**([N/x](λx•P)) ≡ λy•P.
- **subst**([N/x](λy•P)) ≡ λy•**subst**([N/x]P)

if x ≠ y and y is not free in N or x is not free in P.

- **subst**([N/x](λy•P)) ≡λz•**subst**([N/z]**subst**([z/y]P))
  if y ≠ x and y is free in N and x is free in P
  (where z is not free in (N P)).

A.5.4 α–renaming and β–reduction

- α–renaming: λx•M
  If x y are distinct variables then replacing x by y in λx•M results in λy•**subst**([y/x]M): We can rename the formal parameter of a λ-function expression provided that no free variables of its body M thereby become bound.
- β–reduction: (λx•M)(N)
  All free occurrences of x in M are replaced by the expression N provided that no free variables of N thereby become bound in the result.
  (λx•M)(N) ≡ **subst**([N/x]M)

A.5.5 Function signatures

For some functions, we want to abstract from the function body:

```
value
  obs_Pos_Aircraft: Aircraft → Pos,
  move: Aircraft × Dir → Aircraft,
```

A.5.6 Function definitions

Functions — with body — can be defined explicitly:

value
  f: A × B × C → D
  f(a,b,c) ≡ Value_Expr

  g: B-**infset** × (D $\xrightarrow[m]{}$ C-**set**) $\xrightarrow{\sim}$ A*
  g(bs,dm) ≡ Value_Expr
  pre $\mathcal{P}$(dm)

or implicitly:

value
  f: A × B × C → D
  f(a,b,c) **as** d
  post $\mathcal{P}_1$(d)

  g: B-**infset** × (D $\xrightarrow[m]{}$ C-**set**) $\xrightarrow{\sim}$ A*
  g(bs,dm) **as** al
  pre $\mathcal{P}_2$(dm)
  post $\mathcal{P}_3$(al)

The symbol $\xrightarrow{\sim}$ indicates that the function is partial and thus not defined for all arguments. Partial functions should be assisted by pre–conditions stating the criteria for arguments to be meaningful to the function.

A.5.7 Let expressions

Simple (i.e., non–recursive) **let** expressions:

let a = $\mathcal{E}_d$ **in** $\mathcal{E}_b$(a) **end**

is an "expanded" form of:

$$(\lambda \mathrm{a}.\mathcal{E}_b(\mathrm{a}))(\mathcal{E}_d)$$

Recursive **let** expressions are written as:

let f = λa:A • E(f) **in** B(f,a) **end**

is "the same" as:

let f = **Y**F **in** B(f,a) **end**

where:

F ≡ λg•λa•(E(g)) and YF = F(YF)

Predicative **let** expressions:

let a:A • $\mathcal{P}$(a) **in** $\mathcal{B}$(a) **end**

express the selection of a value a of type A which satisfies a predicate $\mathcal{P}$(a) for evaluation in the body $\mathcal{B}$(a).

Patterns and *Wild Cards* can be used:

let {a} ∪ s = set **in** ... **end**
let {a,_} ∪ s = set **in** ... **end**

let (a,b,...,c) = cart **in** ... **end**
let (a,_,...,c) = cart **in** ... **end**

```
let ⟨a⟩⌢ℓ = list in ... end
let ⟨a,_,b⟩⌢ℓ = list in ... end

let [ a↦b ] ∪ m = map in ... end
let [ a↦b,_ ] ∪ m = map in ... end
```

A.5.8 Applicative conditionals

Various kinds of conditional expressions are offered by RSL:

```
if b_expr then c_expr else a_expr end

if b_expr then c_expr end ≡ /∗ same as: ∗/
   if b_expr then c_expr else skip end

if b_expr_1 then c_expr_1
elsif b_expr_2 then c_expr_2
elsif b_expr_3 then c_expr_3
...
elsif b_exprt_n then c_expr_n end

case expr of
   choice_pattern_1 → expr_1,
   choice_pattern_2 → expr_2,
   ...
   choice_pattern_n_or_wild_card → expr_n
end
```

A.5.9 Common operator/operand constructs

```
⟨Expr⟩ ::=
          ⟨Prefix_Op⟩ ⟨Expr⟩
        | ⟨Expr⟩ ⟨Infix_Op⟩ ⟨Expr⟩
```

| ⟨Expr⟩ ⟨Suffix_Op⟩
| ...
⟨Prefix_Op⟩ ::=
− | ∼ | ∪ | ∩ | **card** | **len** | **inds** | **elems** | **hd** | **tl** | **dom** | **rng**
⟨Infix_Op⟩ ::=
= | ≠ | ≡ | + | − | ∗ | ↑ | / | < | ≤ | ≥ | > | ∧ | ∨ | ⇒
| ∈ | ∉ | ∪ | ∩ | \ | ⊂ | ⊆ | ⊇ | ⊃ | ⌢ | † | ∘
⟨Suffix_Op⟩ ::= !

A.6 Imperative constructs

Often, following the RAISE method, software development starts with highly abstract–applicative which, through stages of refinements, are turned into concrete and imperative. Imperative constructs are thus inevitable in RSL.

A.6.1 Variables and assignment

0. **variable** v:Type := expression
1. v := expr

A.6.2 Statement sequence and skip

Sequencing is done using the ';' operator. **skip** is the empty statement having no value or side–effect.

2. **skip**
3. stm_1;stm_2;...;stm_n

A.6.3 Imperative conditionals

4. **if** expr **then** stm_c **else** stm_a **end**
5. **case** e **of**: p_1→S_1(p_1),...,p_n→S_n(p_n) **end**

A.6.4 Iterative conditionals

6. **while** expr **do** stm **end**
7. **do** stmt **until** expr **end**

A.6.5 Iterative sequencing

8. **for** b **in** list_expr • P(b) **do** S(b) **end**

A.6.6 Process channels

Let A, B and KIdx stand for a type of (channel) messages, respectively; then:

channel c:A
channel { k[i]:B • i:KIdx }

declare a channel, c, and a set of channels, k[i], able of communicating values of the designated types.

A.6.7 Process composition

Let P and Q stand for names of process functions, i.e., of functions which express willingness to engage in input and/or output events, thereby communicating over declared channels.

Let P() and Q(i) stand for process expressions, then:

```
P() ‖ Q(i)    Parallel composition
P() [] Q(i)   Non−−deterministic External Choice (either/or)
P() ⌈⌉ Q(i)   Non−−deterministic Internal Choice (either/or)
```

express the parallel of two processes, respectively the non–deterministic choice between two processes: Either external or internal.

A.6.8 Input/Output processes

Let c, k[i] and e designate a channels of type A and B, respectively; then:

```
c ?, k[i] ?      Input
c ! e, k[i] ! e  Output
```

expresses the willingness of a process to engage in an event that reads an input, and respectively writes an output.

A.6.9 Process signatures and definitions

The below signatures are just examples. They emphasise that process functions must somehow express, in their signature via which channels they wish to engage in input and output events.

```
value
  P: Unit → in c out k[i] Unit
  Q: i:KIdx → out c in k[i] Unit

  P() ≡ ... c ? ... k[i] ! e ...
  Q(i) ≡ ... k[i] ? ... c ! e ...
```

The process function definitions (i.e., their bodies) express possible events.

A.6.10 Simple RSL specifications

Often, we do not want to encapsulate small specifications in schemes, classes, and objects; as often done in RSL. Not using schemes, classes, nor objects (see [134, 135]), an RSL specification is simply a sequence of one or more types, values (including functions), variables, channels and axioms:

```
type
  ...
variable
  ...
channel
  ...
value
  ...
axiom
  ...
```

Bibliography

[1] Martín Abadi and Luca Cardelli. *A Theory of Objects*. Springer, 1996.

[2] Christoffer Alexander and Barry Poyner. The atoms of environmental structure. In N. Cross, editor, *Developments in Design Methodology*, chapter 2.2, pages 123–133. John Wiley & Sons, 1984.

[3] Christopher Alexander, S. Ishikawa, and M. Silverstein. *A Pattern Language*, volume 2 of *Center for Environmental Structure Series*. Oxford University Press, New York, NY, 1977.

[4] Guillermo Arango. DOMAIN ANALYSIS — from art form to engineering discipline. *ACM*, 1989.

[5] Guillermo Arango. A brief introduction to domain analysis. *ACM*, 1994.

[6] Geir Arngrímsson and Johan Vesterager. Overblik over standarden ISO-STEP samt erfaringer fra konkret anvendelse af standarden. Technical report, Department of process engineering, Technical University of Denmark, 1991.

[7] Alessandro Artale, Enrico Franconi, Nicola Guarino, and Luca Pazzi. Part-whole relations in object-centered systems: An overview. *Data & Knowledge Engineering*, 20(3):347–383, 1996.

[8] Robert Audi, editor. *Cambridge Dictionary of Philosophy*. Cambridge University Press, second edition, 1999.

[9] Ralph-Johan Back. Correctness Preserving Program Refinements: Proof Theory and Applications. *Mathematical Center Tracts.*, 131, 1980.

[10] Ralph-Johan Back, Abo Akademi, J. von Wright, and F. B. Schneider. *Refinement Calculus: A Systematic Introduction*. Springer–Verlag New York, Inc., 1998.

[11] Mark Balaguer. *Platonism and Anti–Platonism in Mathematics*. Oxford University Press, 1998.

[12] François Bancilhon and Setrag Khoshafian. A calculus for complex objects. *Journal of Computer and Systems Sciences*, 38, 1989.

[13] Garrett Birkhoff. *Lattice Theory*, volume 25. American Mathematical Society Colloquium Publications, Providence, Rhode Island, third edition, 1995.

[14] Dines Bjørner. Software systems engineering — from domain analysis to requirements capture: An air traffic control example. In *2nd Asia-Pacific Software Engineering Conference (APSEC '95)*. IEEE Computer Society, 6–9 December 1995. Brisbane, Queensland, Australia.

[15] Dines Bjørner. Domains as a prerequisite for requirements and software: Domain perspectives and facets, requirements aspects and software views. In *Proceedings US DoD/ONR Workshop, Bernried*, October 1997.

[16] Dines Bjørner. A Triptych Software Development Paradigm: Domain, Requirements and Software. Towards a Model Development of A Decision Support System for Sustainable Development. In *Festschrift for Hans Langmaack: Correct Systems Design: Recent Insight and Advances*, volume 1710 of Lecture Notes in Computer Science, pages 29–60. Springer–Verlag, October 1999.

[17] Dines Bjørner. Civil engineering specification languages enriching a classical engineering field with computing science concepts, 1999. Unpublished.

[18] Dines Bjørner. P^3imcacs: A project/production planning, information, monitoring, control & communication system. Technical report, Department of Computer Science, Technical University of Denmark, 1999.

[19] Dines Bjørner. Informatics: A truly interdisciplinary science – prospects for an emerging world. In *Proc. AIT, Bangkok, Thailand; invited paper*, pages 71–84, august 2000.

[20] Dines Bjørner. Towards the E–Market: To understand the E–Market we must first understand "The Market". In *Government E–Commerce Development*. Ningbo Science & Technology Commission. Ningbo, Zhejian Province, China, April 2001.

[21] Dines Bjørner. Domain Models of "The Market" - in Preparation for E–Transaction Systems. In Haim Kilov and Ken Baclawski, editors, *Practical Foundations of Business and System Specifications*. Kluwer Academic Press, 2002.

[22] Dines Bjørner. An Ontology for a TripTych Formal Software Development Method. In *Radical Innovations for Systems and Software Engineering, The Monterey Workships. Venice, Italy*. Springer–Verlag, October 2002.

[23] Dines Bjørner. Domain engineering: A "radical innovation" for systems and software engineering ? In *Verification: Theory and Practice*, volume 2772 of *Lecture Notes in Computer Science*, Heidelberg, October 7–11 2003. Springer–Verlag. The Zohar Manna International Conference, Taormina, Sicily 29 June – 4 July 2003.

[24] Dines Bjørner. Financial service institutions: Banks, securities trading, insurance, *&c.* towards a domain theory for work flow systems. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark, 2003. This paper is one of a series of papers currently being submitted for publication in [30].

[25] Dines Bjørner. Health–care systems. towards a domain theory for work flow systems. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark, 2003. This paper is one of a series of papers currently being submitted for publication in [30].

[26] Dines Bjørner. Logistics. towards a domain theory for work flow systems. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark, 2003. This paper is one of a series of papers currently being submitted for publication in [30].

[27] Dines Bjørner. Models, semiotics, documents and descriptions: Towards a software engineering literacy. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark, 2003. This paper is one of a series of papers currently being submitted for publication in [30].

[28] Dines Bjørner. Projects & production: Planning, plans & execution. towards a domain theory for work flow systems. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark, 2003. This paper is one of a series of papers currently being submitted for publication in [30].

[29] Dines Bjørner. "What is a Method ?" An Essay on Some Aspects of Software Engineering. *Programming methodology. Monographs in Computer Science*, pages 175–203, 2003.

[30] Dines Bjørner. *Software Engineering*, volume Vol. 1: Abstraction and Modelling, Vol. 2: Advanced Specification Techniques, Vol. 3: From Domains via Requirements to Software, Vol. 4: Management. Springer–Verlag, 2004–2005. Volumes 1–3 to be published either late 2004 or early 2005; Volume 4 planned.

[31] Dines Bjørner, Chris George, and Søren Prehn. Computing systems for railways — a rôle for domain engineering. invited paper. In *6th World Conference on Integrated Design & Process Technology, Pasadena, California, USA*, June 2002.

[32] Kurt Löwnertz Bo-Christer Björk and Arto Kiviniemi. ISO DIS 13567 — the proposed international standard for structuring layers in computer aided building design. Technical report, ITcon, 1997.

[33] Grady Booch, Jim Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison–Wesley, 1998.

[34] Chris Brink, Katarina Britz, and Renate A. Schmidt. Peirce algebras. *Formal Aspects of Computing*, 6:339–358, 1996.

[35] BSAB. BSAB 96, the Swedish construction industry classification system. Technical report, The Swedish Building Centre, Svensk Byggtjänst, 1999.

[36] Mario Bunge. *Semantics I: Sense and Reference*, volume 1 of *Treatise on Basic Philosophy*. Reidel, Dordrecht and Boston, 1974.

[37] Mario Bunge. *Semantics II: Interpretation and Truth*, volume 2 of *Treatise on Basic Philosophy*. Reidel, Dordrecht and Boston, 1974.

[38] Mario Bunge. *Ontology I: The Furniture of the World*, volume 3 of *Treatise on Basic Philosophy*. Reidel, Dordrecht and Boston, 1977.

[39] Mario Bunge. *Ontology II: A World of Systems*, volume 4 of *Treatise on Basic Philosophy*. Reidel, Dordrecht and Boston, 1979.

[40] Mario Bunge. *Epistemology and Methodology I: Exploring the World*, volume 5 of *Treatise on Basic Philosophy*. Reidel, Dordrecht and Boston, 1983.

[41] Keith Campbell. The metaphysics of abstract particulars. In D.H.Mellor and Alex Oliver, editors, *Properties*, Oxford Readings in Philosophy. Oxford University Press, 1997.

[42] Luca Cardelli. A Semantics of Multiple Inheritance. *Information and Computation*, 76(2/3):138–164, 1988.

[43] Wolfgang Carl. Frege — A Platonist or a Neo–Kantist? In Albert Newen, Ulrich Nortmann, and Rainer Stuhlmann-Laeisz, editors, *Building on Frege*. CSLI Publications. Center for the Study of Language and Information. Stanford, California, 2001.

[44] Jaelson Castro, Manuel Kolp, and John Mylopoulos. Towards requirements–driven information systems engineering: The *Tropos* project. *Information Systems*, 2002.

[45] Edith Cherry. *Programming for Design : From Theory to Practice*. John Wiley & Sons, November 1998.

[46] Alonzo Church. *Introduction to Mathematical Logic*. Princeton University Press, 1956.

[47] Nino B. Cocchiarella. *Axiomathes*, chapter Logic and ontology, pages 117–150. Kluwer Academic Publishers, 2001.

[48] Ingetraut Dahlberg. A referent–oriented, analytical concept theory for INTERCONCEPT. In *Intern. Classificat.*, volume 5 of *3*, 1978.

[49] Chris Daly. Tropes. In D.H.Mellor and Alex Oliver, editors, *Properties*, Oxford Readings in Philosophy. Oxford University Press, 1997.

[50] Anne Dardenne, Axel Van Lamsweerde, and Stephen Fickas. Goal–directed requirements acquisition. *Science of Computer Programming*, 20:3–50, 1993.

[51] Donald Davidson. Truth and meaning. In Oxford Readings in Philosophy, editor, *Meaning and Reference*, pages 92–110. Oxford University Press, 1993.

[52] Marcel de Waard. *Computer Aided Conformance Checking*. Marcel de Waard, 1992.

[53] G. Dedene and M. Snoeck. A Model–driven Entity–Relationship Object–oriented Development. *ACM SIGSOFT Software Engineering notes*, 19(3), 1994.

[54] Nachum Dershowitz and Jean-Pierre Jouannaud. 6 rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Methods and Semantics, pages 243–320. North–Holland, Amsterdam, 1990.

[55] D.H.Mellor and Alex Oliver. Introduction. In D.H.Mellor and Alex Oliver, editors, *Properties*, Oxford Readings in Philosophy. Oxford University Press, 1997.

[56] R. Drogemuller and J. D. Smith. Cooperating Systems for Design and Manufacture: STEPping out with Prolog. In *Proc.of the Third International Conference on the Practical Application of Prolog*, pages 213–224, Paris, 1995.

[57] Robin Drogemuller. Modelling static and dynamic knowledge during design. In *CIB W78 Stanford Conference Programme, Information Technology in Construction*, 1995.

[58] Michael Dummett. *Frege. Philosophy of language.* Harvard University Press, 1991.

[59] Michael Dummett. Frege's distinction between sense and reference. In Oxford Readings in Philosophy, editor, *Meaning and Reference.* Oxford University Press, 1993.

[60] Charles M. Eastman. *Building Product Models: Computer Environments Supporting Design and Construction.* CRC Press, 1999.

[61] Asger Eir. A semi–semantic document system for construction specifications. Master's thesis, Department of Information Technology, Technical University of Denmark, 2000.

[62] Asger Eir and Anders Ekholm. From rough to final designs by incremental set–inclusion of properties. In *eWork and eBusiness in Architecture, Engineering, Construction, proceedings of the ECPPM 2002*, pages 293–300. Swets & Zeitlinger Publishers, 2002.

[63] Anders Ekholm. *Systemet människa–byggnadsverk — ett ontologiskt perspektiv (The System man–building — an ontological view).* Report, The Swedish National Council for Building Research, Stockholm, Sweden, 1987. R22.

[64] Anders Ekholm. A conceptual framework for classification of construction works. *IT con*, 1, 1995.

[65] Anders Ekholm. Information systems for architectural design — Experiences from the BAS•CAAD and ACTIVITY projects. *Nordic Journal of Architectural Research*, 3:79–86, 2001.

[66] Anders Ekholm and Sverker Fridqvist. Object–oriented caad — design object structure and models for buildings, user organisation and site. In M. Fisher, K. Law, and B. Luiten, editors, *Modeling of buildings through their life cycle, CIB W78/TG10 workshop on computers and information in construction*, pages 553–564. Stanford University, Ca, USA, August 1995.

[67] Anders Ekholm and Sverker Fridqvist. A dynamic information system for design applied to the construction context. In B-C. Björk and A. Jägbeck, editors, *Proceedings of the CIB W78 workshop The life–cycle of Construction IT*, pages 219–232, June 1998.

[68] Anders Ekholm and Sverker Fridqvist. The BAS•CAAD information system for design — principles, implementation, and a design scenario. In Godfried Augenbroe and Charles Eastman, editors, *Computers in Building. Proceedings of the Eighth International Conference on Computer Aided Architectural Design Futures, CAADfutures'99*, pages 149–164, Atlanta, 7-8 June 1999. Kluwer Academic Publishers.

[69] Anders Ekholm and Sverker Fridqvist. A concept of space for building classification, product modelling, and design. *Automation in Construction*, 9:315–328, 2000.

[70] Anders Ekholm, Lars Häggström, Väino Tarandi, and Olle Thåström. Application of ifc in sweden — phase 2, final report. Technical report, The Swedish Building Centre, 2000.

[71] Gottlob Frege. Function and concept. In D.H.Mellor and Alex Oliver, editors, *Properties*, Oxford Readings in Philosophy, pages 34–44. Oxford University Press, 1891.

[72] Gottlob Frege. On sense and reference. In A. W. Moore, editor, *Meaning and Reference*, Oxford Readings in Philosophy, pages 23–42. Oxford University Press, 1892.

[73] Gottlob Frege. Letter to jourdain (originally extract from undated letter). In Oxford Readings in Philosophy, editor, *Meaning and Reference*. Oxford University Press, 1993.

[74] Gottlob Frege. On concept and object (Über begriff und gegenstand). In *Funktion und Begriff*. Vandenhoeck & Ruprecht in Göttingen, 1994.

[75] Sverker Fridqvist. *Property–Oriented Information Systems for Design*. PhD thesis, Division of Architectural and Building Design Methods, Lund University, 2000.

[76] Yoshihiko Futamura. Partial evaluation of computation process — an approach to a compiler–compiler. In *Systems.Computers.Controls*, volume 2, No.5, 1971. Also published in Higher–Order and Symbolic Computation 12, p. 381–391 (1999), Kluwer Academic Publishers, 2000.

[77] Yoshihiko Futamura, Kenroku Nogi, and Akihiko Takano. Essence of generalized partial computation. In *Theoretical Computer Science*, volume 90, 1991.

[78] Per Galle. Computer methods in architectural problem solving: Critique and proposals. In *CAAD: Education - Research and Practice (eCAADe)*, pages 6.4.1–6.4.21, 1989.

[79] Per Galle. Artefact specification, design, and production as a process of communication. In G. E. Lasker, editor, *Advances in Systems Research and Cybernetics, Proceedings of the InterSymp-99*, volume III, pages 58–62. Windsor, International Institute for Advanced Studies in Systems Research and Cybernetics, August 1999.

[80] Per Galle. Design as intentional action: a conceptual analysis. *Design Studies*, 20(1):57–81, January 1999.

[81] Per Galle. Data, model, and reality. In *InterSymp–2000*, 2000.

[82] Per Galle. Generating concept representations from examples using set–based notation. In *Advances in Computer-Based and Web-Based Collaborative Systems, InterSymp–2000*, pages 27–33, 2000.

[83] Per Galle. Ontological backdrop. April 2000.

[84] Per Galle. *Objects of design (in preparation).* 2004.

[85] Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis.* Springer, 1999.

[86] Peter Gärdenfors. *Conceptual Spaces.* The MIT Press, 2000.

[87] J. H. Garrett, Jr. and M. Maher Hakim. Class-centered versus object-centered approaches for modeling engineering design information. In *Proceedings of the IKM-Colloquium on Mathematics and Information Sciences in Building Engineering*, pages 267–272, March 1994.

[88] Peter Geach. *God and the Soul.* St. Augustine's Press, second edition, 2002.

[89] W. F. Gielingh. General Reference Model for AEC Product Definition Data (version 3). Technical Report 68.5.4201, Instituut TNO voor Bouwmaterialen en Bouwconstructies (IBBC), 1987.

[90] L. M. Giertz. Sfb and its development 1950–1980. Technical report, CIB/SfB, 1982.

[91] Nicola Guarino. The ontological level. 16^{TH} *Wittgenstein Symposium*, 1995. Revised version.

[92] Nicola Guarino. Understanding, building, and using ontologies: A commentary to using explicit ontologies in kbs development. In *International Journal of Human and Computer Studies*, pages 293–310, 1997.

[93] Nicola Guarino and Christopher Welty. Identity, unity, and individuality: Towards a formal toolkit for ontological analysis. Technical report, LADSEB/CNR, February 2000.

[94] Nicola Guarino and Christopher A. Welty. Ontological analysis of taxonomic relationships. In *International Conference on Conceptual Modeling / the Entity Relationship Approach*, pages 210–224, 2000.

[95] Alain Guénoche and Iven van Mechelen. Galois approach to the induction of concepts. In *Categories and Concepts: Theoretical views and inductive Data Analysis*, 1993.

[96] Hele-Mai Haav. An object classifier based on galois approach. In H. Kangassalo and others., editors, *Proceedings of Information Modelling and Knowledge Bases, VIII*, 1997.

[97] M. Maher Hakim and J. H. Garrett, Jr. Modeling engineering design information: An object-centered approach. In *Proceedings of the First ASCE Congress on Computing in Civil Engineering*, pages 563–571, June 1994.

[98] Anne E. Haxthausen and Jan Peleska. Formal Development and Verification of a Distributed Railway Control System. In *World Congress on Formal Methods (2)*, pages 1546–1563, 1999.

[99] Ann Hendricx. Shape, Space and Building Element: Development of a Conceptual Object Model for the Design Process. In *Proceedings of the 15th European Conference on Education in Computer Aided Architectural Design in Vienna, Austria*, September 1997.

[100] Risto Hilpinen. Artifact. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. The Metaphysics Research Lab. Stanford, CA 94305-4115, Spring 1999.

[101] Tu Bao Ho. Acquiring concept approximations in the framework of rough concept analysis. In *7th European-Japanese Conference on Information Modelling and Knowledge Bases, Toulouse*, 1997.

[102] Douglas R. Hofstadter. *Gödel, Escher, Back: an Eternal Golden Braid*. Basic Books, Inc., 1979.

[103] Michael Ingleby. Galois connections in railway formalisation. In *Proceedings of the fourth FMERail Workshop*, 1999.

[104] Niel Iscoe, Gerald B. Williams, and Guillermo Arango. Domain modeling for software engineering. *IEEE*, 1991.

[105] Frank Jackson. Statements about universals. In D.H.Mellor and Alex Oliver, editors, *Properties*, Oxford Readings in Philosophy. Oxford University Press, 1997.

[106] Michael Jackson. Problems, methods and specialisation. *Software Engineering Journal*, November 1994.

[107] Michael Jackson. *Software Requirements & Specifications — a lexicon of practice, principles and prejudices.* Addison–Wesley, 1995.

[108] Michael Jackson. The meaning of requirements. *Annals of Software Engineering*, 3:5–21, 1997.

[109] Michael Jackson and Pamela Zave. Domain descriptions. In *IEEE International Symposium on Requirements Engineering*, pages 56–64. IEEE CS Press, 1993.

[110] Kim Jacobsen. *KOSA–modellen.* PhD thesis, Department of Planning, Technical University of Denmark, 1997.

[111] G. G. Jensen and K. Olsen, editors. *Teknisk Ståbi (technical reference guide).* Teknisk Forlag, Denmark, 1991.

[112] J.L.Austin. *How To Do Things With Words.* Oxford University Press, second edition, 1976.

[113] Neil D. Jones, Carsten K. Gomard, and Peter Sestoft. *Partial Evaluations and Automatic Program Generation.* Originally Prentice Hall International, now a web–document, 1999.

[114] Michael Jubien. *Contemporary Metaphysics An Introduction.* Contemporary Philosophy. Blackwell Publishers, 1997.

[115] Anthony Kenny. *Frege: An introduction to the Founder of Modern Analytic Philosophy.* Blackwell, 2000.

[116] A. R. Lacey. *A Dictionary of Philosophy.* Routledge, third edition, 2000.

[117] A. Lambsweerde, A. Dardenne, and F. Dubisy. The KAOS Project: Knowledge acquisition in automated specification of software. In *Proceedings of the AAAI Spring S Series, Stanford University*, March 1991.

[118] Emmanuel Letier and Axel van Lamsweerde. Agent–Based Tactics for Goal–Oriented Requirements Elaboration. In *ICSE'2002, 24th International Conference on Software Engineering.* ACM Press, May 2002.

[119] David Lewis. *Counterfactuals.* Blackwell Publishers, 1973.

[120] David Lewis. *On the Plurality of Worlds.* Blackwell Publishers, 1986.

[121] David Lewis. Modal realism at work: Properties. In D.H.Mellor and Alex Oliver, editors, *Properties*, Oxford Readings in Philosophy. Oxford University Press, 1997.

[122] David Lewis. New work for a theory of universals. In D.H.Mellor and Alex Oliver, editors, *Properties*, Oxford Readings in Philosophy, pages 188–228. Oxford University Press, 1997.

[123] Thomas Liebich and Jeffrey Wix. Industry Foundation Classes – Release 2x, IFC Technical Guide. Technical report, IAI, October 2000.

[124] Bertrand Meyer. *Object-Oriented Software Construction*. Prentice-Hall, Inc., 1988.

[125] Zbigniew Michalewicz and David B. Fogel. *How to solve it: Modern heuristics*. Springer–Verlag Berlin Heidelberg 2000, third edition, 2002.

[126] Carroll Morgan, Ken Robinson, and Paul Gardiner. On the Refinement Calculus. Technical monograph, Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford, UK, October 1988.

[127] H. R. Nielson and F. Nielson. *Semantics with Applications: A Formal Introduction*. Wiley, 1992.

[128] Jørgen Fischer Nilsson. An algebraic logic for concept structures. *Information Modelling and Knowledge Bases V*, pages 75–84, 1993.

[129] Jørgen Fischer Nilsson. A concept object algebra $\mathrm{ca}^{+\times[:]}$. In *Proceedings from the 2nd. European–Japanese Seminar on Information Modeling and Knowledge Bases*. IOS Press, Amsterdam, 1993.

[130] Jørgen Fischer Nilsson and Hele-Mai Haav. Inducing queries from examples as concept formation. In *8th European Japanese Conference on Information Modelling and Knowledge Bases*, 1998.

[131] Jørgen Fischer Nilsson and Jari Palomäki. Towards Computing with Extensions and Intensions of Concepts. In *7th European–Japanese Conference on Information Modelling and Knowledge Bases*, pages 100–114, 1997.

[132] Andreas L. Opdahl and Brian Henderson-Sellers. Ontological evaluation of the UML using the Bunge–Wand–Weber model. *Software and Systems Modelling (SoSyM)*, pages 43–67, 2002.

[133] W. V. Quine. On what there is. In D.H.Mellor and Alex Oliver, editors, *Properties*, Oxford Readings in Philosophy. Oxford University Press, 1997.

[134] The RAISE Language Group. *The RAISE Specification Language*. Prentice Hall, CRI A/S, 1992.

[135] The RAISE Method Group. *The RAISE Development Method*. Prentice Hall, CRI A/S, 1995.

[136] F. P. Ramsey. Universals. In D.H.Mellor and Alex Oliver, editors, *Properties*, Oxford Readings in Philosophy. Oxford University Press, 1997.

[137] The Øresund Consortium. *Contract No. 3 – Bridges. Contract document No.1: Contract Agreement*. The Øresund Bridge Consortium, November 1995.

[138] Yacine Rezgui, Grahame Cooper, Bo-Christer Björk, and Jean-Christophe Escudie. From construction product information to consistent project documentation: the condor approach. In Robin Drogemuller and James Cook, editors, *Proceedings of Information Technology Support for Construction Process Reengineering*. CIB, July 1997.

[139] N. F. M. Roozenburg and J. Eekels. *Product Design: Fundamentals and Methods*, chapter What is design? John Whiley & Sons, 1995.

[140] Bertrand Russell. On denoting. *Mind 14*, pages 479–493, 1905.

[141] Bertrand Russell. Descriptions. In Oxford Readings in Philosophy, editor, *Meaning and Reference*. Oxford University Press, 1993.

[142] Bertrand Russell. On our knowledge of universals. In D.H.Mellor and Alex Oliver, editors, *Properties*, Oxford Readings in Philosophy, pages 51–56. Oxford University Press, 1997.

[143] Bertrand Russell. The world of universals. In D.H.Mellor and Alex Oliver, editors, *Properties*, Oxford Readings in Philosophy, pages 45–50. Oxford University Press, 1997.

[144] R. M. Sainsbury. Sense Without Reference. In Albert Newen, Ulrich Nortmann, and Rainer Stuhlmann-Laeisz, editors, *Building on Frege*. CSLI Publications. Center for the Study of Language and Information. Stanford, California, 2001.

[145] Frank Salisbury. *Briefing Your Architect*. Architectural Press, 1998.

[146] Geuorgui Satchok. Metropolitan in–street on–route passenger transport: Monitoring and control. Technical Report 110, UNU/IIST, June 1997.

[147] Preben Scheutz. *Projektstyring & byggeri*. Teknisk Forlag A/S, 1988.

[148] D. A. Schmidt. *Denotational Semantics: a Methodology for Language Development*. Allyn & Bacon, Inc., 1986.

[149] Donald A. Schön. *The Reflective Practitioner*. Ashgate Publishing Group, 1983.

[150] John R. Searle. *Speech Act*. CUP, 1969.

[151] Stewart Shapiro. *Philosophy of Mathematics — structure and ontology*. Oxford University Press, 1997.

[152] Sidney Shoemaker. *Identity, cause, and mind*. Cambridge University Press, 1984.

[153] Sydney Shoemaker. Causality and properties. In D.H.Mellor and Alex Oliver, editors, *Properties*, Oxford Readings in Philosophy, pages 228–254. Oxford University Press, 1997.

[154] Herbert A. Simon. *The Sciences of the Artificial*. MIT Press, Cambridge, 1981.

[155] Peter Simons. *Parts: A Study in Ontology*. Clarendon Press — Oxford, 1987.

[156] Ira A. Smith and Philip R. Cohen. Toward a semantics for a speech act based agent communications language. In Tim Finin and James Mayfield, editors, *Proceedings of the CIKM '95 Workshop on Intelligent Information Agents*, Baltimore, Maryland, 1995.

[157] Ernest Sosa and Michael Tooley. Introduction. In Ernest Sosa and Michael Tooley, editors, *Causation*, Oxford Readings in Philosophy. Oxford University Press, 1993.

[158] John F. Sowa. *Knowledge Representation — Logical, Philosophical, and Computational Foundations*. Brooks/Cole, 2000.

[159] A/S Storebælt. Foreløbig teknisk–økonomisk vurdering af en fast forbindelse over femern bælt, 1990.

[160] Alfred Tarski. The Semantic Conception of Truth and the Foundations of Semantics. *Philosophy and Phenomenological Research*, 4, 1944.

[161] R. D. Tennent. *Semantics of programming languages*. Prentice Hall International (UK) Ltd, 1991.

[162] T.M.Frose and Jr. B.C.Paulson. Integrating project management systems through shared object–oriented project models. In D.E.Grierson, G.Rzevski, and R.A.Adey, editors, *Applications of Artificial Intelligence in Engineering*, volume VII, 1992.

[163] Z. Turk and B. C. Bjork. Document management systems as an integral step towards CIC. In *Workshop on Computer Integrated Construction*. CIB W78, August 1994.

[164] Rianne Valkenburg and Kees Dorst. The reflective practice of design teams. *Design Studies*, 19:249–271, 1998.

[165] Hung Dang Van, Chris George, Tomasz Janowski, and Richard Moore, editors. *Specification Case Studies in RAISE*, chapter Formalising Production Processes. FACIT. Springer, 2002.

[166] Axel van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *RE'01 - 5th IEEE International Symposium on Requirements Engineering, Toronto, August, Invited Paper*, pages 249–263, 2001.

[167] Jos van Leeuwen. *Modelling Architectural Design Information by Features, and approach to dynamic product modelling for application in architectural design*. PhD thesis, Eindhoven University of Technology, The Netherlands, 1999.

[168] Joakim von Wright. The lattice of data refinement. *Acta Informatica*, 31(2):105–135, 1994.

[169] Žiga Turk. On theoretical backgrounds of cad. In I.Smith, editor, *Structural Engineering Applications of Artificial Intelligence, Lecture Notes in Artificial Intelligence 1454*, pages 490–496. Springer, Berlin, 1998.

[170] Žiga Turk. Phenomenologial foundations of conceptual product modelling in aec. *International Journal of AI in Engineering*, 15:83–92, 2001.

[171] Donald C. Williams. On the elements of being: I. In D.H.Mellor and Alex Oliver, editors, *Properties*, Oxford Readings in Philosophy. Oxford University Press, 1997.

[172] Crispin Wright. Wittgenstein's rule–following considerations and the central project of theoretical linguistics. In Alexander George, editor, *Reflections on Chomsky*. Blackwell, 1989.

[173] Eric Yu. Towards Modelling and Reasoning Support for Early–Phase Requirements Engineering. In *3rd IEEE Int. Symp. on Requirements Engineering (RE'97)*, pages 226–235. IEEE, January 1997.

[174] Eric Yu and John Mylopoulos. Using Goals, Rules and Methods to Support Reasoning in Business Process Reengineering. *International Journal of Intelligent Systems in Accounting, Finance, and Management*, 5(1):1–13, March 1996.

Index