USING DESIGN LANGUAGES FOR CONCEPTUAL MODELLING: THE UML CASE

by

JOERG MAGNUS EVERMANN

Diplom Wirtschafts-Informatik
Westfälische Wilhelms Universität Münster, 1998

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE STUDIES
(Management Information Systems)

We accept this thesis as conforming
to the required standard

.....

.....

.....

THE UNIVERSITY OF BRITISH COLUMBIA

November 2003

Abstract

Information systems are representations of and situated in the business and organization. In order to develop effective information systems (IS), the first step must be to understand and describe this real world domain. System analysis is this first step, the final result of which is the conceptual model, a formal description of the business and organizational domain. It serves as the communication medium to further common domain understanding. System design, the second step in IS development, builds on this understanding, together with other requirements regarding e.g. functionality, performance, quality, usability, to design the software system. The conceptual model serves as input to this phase.

Every model must be expressed in a language. However, there exists no widely accepted language for conceptual modelling of the business or the organization. On the other hand, recent years have seen the emergence and wide acceptance of object-oriented languages in general, and the Unified Modelling Language (UML) specifically, for IS design.

This study examines the suitability of using such design languages for conceptual modelling. In order for a language to be usable for modelling business and organizational domains, the language constructs must possess real-world semantics, i.e. it must be clear what they refer to in the real world, not only in the software domain. Based on ontology, the branch of philosophy that deals with what exists in the real world, this study assigns such meaning to UML constructs. Based on these semantics, ontological assumptions are used to derive modelling rules for UML when UML is used for conceptual modelling. These rules are formalized using the UML meta-model.

A case study is conducted which applies the proposed rules in a medium size IS development project and notes their beneficial effects on the analysis process and the final conceptual model. An experimental study is conducted to show specific benefits to domain understanding, induced by models which conform to the proposed rules.

The chosen method of analysis of languages is applicable not only to UML, chosen as an example here, but is generalizable to other languages as well. The results derived in this study, other than the formalization by means of the specific language meta-model, are therefore generalizable to other object-oriented languages.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to acknowledge the invaluable support and advice of my supervisor, Dr. Yair Wand, as well as that of my thesis committee members, Dr. Gail Murphy and Dr. Carson Woo, for this research and the preparation of this thesis. Many thanks go to the university and external examiners for their suggestions.

I am indebted to the Open Source Software community for providing the excellent tools without which this thesis could not have been written. It was prepared using Linux, LaTeX, Ghostscript, Joe, XFig, TCM, R, and ArgoUML among others.

This thesis would not have been possible without the unending support and understanding of my wife Elizabeth throughout the many months that this research was in the making. A very special thank you for the sacrifices she made.

Besonderen Dank an meine Eltern, Josef und Ida, für deren Unterstützung und Verständnis in den letzten vier Jahren, ohne welches diese Arbeit nicht möglich gewesen wäre.

Chapter 1

Introduction

Information systems (IS) are representations of a business or organizational domain: The software and database structures of an inventory IS should reflect the layout of the warehouses with their aisles, shelves and bins. Changes in the data managed by the inventory IS should mirror actual changes of inventory in the warehouse. The structures of a production planning and control (PPC) system should reflect the type of equipment and material on the factory floor. Changes in the data in the PPC system should mirror actual changes of work items and equipment.

Information systems are also situated in and affect the real world domain. The inventory system is used for making decisions about stock levels, purchasing, etc. It is embedded in and affects the business. The production planning and control system is used to make decisions about production schedules, equipment changes, etc. It also is embedded in and affects the business.

For these reasons it is essential that any IS project begin by examining the real-world domain represented and affected by the IS. Hence, the first task in IS development, the analysis phase, is concerned with describing this real world domain through conceptual models. These models are descriptions of the real-world independent of any information systems or information technology aspect: "Conceptual modeling is the activity of formally describing some aspects of the physical and social world around us for purposes of understanding and communication" (Mylopoulos, 1992). The purpose of conceptual models is twofold: (1) To serve as communication medium for understanding of the domain, and (2) to serve as a guide for IS

design (Kung and Solvberg, 1986).

The description of an information system that is developed in the subsequent design phase differs from the initial description of the real world in the analysis phase because the context is increasingly shaped by technical considerations as we progress along the development process. For the IS design phase, the use of object-oriented techniques is well accepted. A number of languages have been proposed (e.g. Booch (1994); Coleman *et al.* (1994); Jacobson (1992)). Several such languages were combined to form the Unified Modeling Language UML (Bezivin and Muller, 1999; OMG, 2001). It has become widely used as a way to describe software elements of an information system during the design phase.

While the difference between analysis and design models, or between business and software models, is widely recognized, the lack of languages specific to conceptual modelling combined with the availability of widely used IS design languages has a number of detrimental effects: (1) Many IS development projects begin without explicitly modelling the real world and developers hold implicit assumptions. (2) Even when the real world is explicated, the use of IS design languages for this task without specific guidance leads developers to confuse aspects of the IS and the real world. For example, an analyst may talk about jobs in the organization as objects, with the already implicit understanding that they will be represented by a specific class in the object-oriented software system. (3) The translation between conceptual and design models is not explicated. This lack of explicit translation can again lead to hidden assumptions made by different stakeholders and developers.

Besides the lack of a widely accepted language, there are other problems potentially affecting IS development. The transition from analysis to design in the system development introduces the following challenges:

- The object of modelling changes, from a real world business domain to the information system artifact.
- The model must increasingly take technical considerations into account.
- The language of description may change.

Clearly, the use of object-oriented languages for both analysis and design, or for conceptual real-world modelling and software modelling, can

solve a number of problems. It can provide a language for conceptual modelling that is already familiar to software designers and so can also be used in the downstream phase of IS design. Thus, no change in language is required, alleviating potential translation problems, inconsistencies and errors introduced by changes in modelling languages. Because of these potential benefits, the central question addressed by this research is:

> Can object-oriented approaches and languages, specifically UML, be used for conceptual modelling and in what way should they be used?

As there is no commonly agreed on language for describing the real world, recent research (Green and Rosemann, 2000; Opdahl and Henderson-Sellers, 1999; Opdahl and Sindre, 1993; Opdahl *et al.*, 1999; Opdahl and Henderson-Sellers, 2001; Parsons and Wand, 1997; Wand and Weber, 1989; Wand *et al.*, 1999) has already investigated the feasibility of *extending* the use of object-oriented IS design languages in general or specific languages such as OML or UML for this purpose. The main problem to overcome is the lack of clear business or organizational meanings for language constructs such as 'object', 'class', 'attribute', 'operation'.

In order to employ a language for the purpose of describing real-world business and organizational domains, we must attach or incorporate real-world semantics. It must be clear which elements of the real world a particular language construct can or cannot refer to. Ontology is the branch of philosophy that deals with what exists in the world (Angeles, 1981) and has been proposed as a baseline against which to evaluate languages for conceptual modelling (Green and Rosemann, 2000; Opdahl and Henderson-Sellers, 1999; Opdahl and Sindre, 1993; Opdahl *et al.*, 1999; Opdahl and Henderson-Sellers, 2001; Parsons and Wand, 1997; Wand and Weber, 1989, 1993; Wand *et al.*, 1999). It can be used to give business meaning, or real world semantics, to a language.

To assign real world semantics to a language, we map ontological concepts to the constructs of the language and vice versa. Besides providing meaning to language elements, these mappings also allow the identification of deficiencies in the language, instances where the mapping is not bijective or ontologically clear. For the meaning of a language and its constructs to be clearest, each language construct should be mapped to exactly one real world concept and vice versa. Thus, a mapping provides maximum ontological clarity if it is bijective (Wand and Weber, 1993). To achieve such clarity,

it may be necessary to map language constructs to ontological concepts only if they appear in a particular context: Suppose that a language provides a construct which, depending on the context in which it is used, expresses two different real-world situations. Ontological clarity can be achieved by proposing appropriate modelling rules which map that construct, when used *in the first context* to the one ontological concept and map it, when used *in the second context*, to the other ontological concept.

Once the ontological semantics are assigned and language constructs are mapped to ontological concepts, the mappings can be used to transfer ontological assumptions to the language. An ontology may suggest that certain situations are possible in the real world while others are not. By virtue of the mappings, some combinations of language elements therefore describe possible real world situations while others describe impossible ones. Formal constraints on the language elements can restrict the set of possible models to allow only the modelling of possible real world situations. These constraints formalize real-world modelling rules in the syntax of a language; they govern the way in which language constructs may be combined to form ontologically valid models.

In summary, thesis is a bottom-up approach that it investigates how object-oriented software design languages can be *extended* to include conceptual modelling.

UML We propose to use UML and formal meta-models in this research. UML serves as an example for object-oriented modelling languages in general. It is used to demonstrate the feasibility and validity of the research approach. UML is chosen for the following reasons:

- UML is the most prominent and widely accepted IS design language. As such, many developers are familiar with it and its graphical language constructs.
- UML is an evolving language. Hence, any research can have practical influences on the evolution of the language standard.
- UML is a language with a well defined syntax and a formal meta-model. Thus, it is unambiguous and it enables the formalization of any research results in terms of this meta-model.
- UML has been developed for modelling software and software components. However, it is not specifically limited to this. Therefore,

it appears possible to extend the application area of UML to include business modelling.

While the Unified Modelling Language has not been developed or intended to be used for the modelling of business and organizations, the advantages of being able to use UML also for conceptual modelling, as well as for software design are manifold. It provides a much-needed formal language for conceptual modelling. It thereby helps alleviate the problems due to lack of language discussed above. Furthermore, a common language can ease the transition from the system analysis stage of the IS development process to the software design stage. As the intended models are valid models in standard UML, they can be understood by system designers. Hence, analysts can use these models not only for communicating amongst themselves for purposes of understanding, but also to communicate with software designers to convey information about the software requirements.

The widespread adoption of UML further enhances the usefulness of the language as a common form of communication among analysts and developers. It makes models immediately understandable to both groups and can thus serve to eliminate costly and error prone translations between two languages. Such translations can be fraught with difficulties as the languages may not be equally expressive or the translation may be very complex and thus errorprone.

The rules and guidelines developed in this thesis are intended to maintain the capacity of the resultant models to serve as the starting points for software design, without requiring elaborate transformation efforts[1]. Hence, the resulting diagrams should be directly translatable to software and programming statements. However, some transformations may possibly be applied to the conceptual model. This may be done for a variety of reasons, for example to increase the computational efficiency of the derived software, to adapt to certain database technologies, to ease the programming effort necessary or to adapt to specificities of the programming language or software framework. However, such implementation driven transformations are beyond the scope of this thesis and must be taken up by further research.

Meta-Models Meta-models are models of a model, they describe the elements used in a model and their relationships. Meta-models thus describe and define the modelling language. Meta-models and formal constraints

[1]This will be examined in Chapters 7 and 10.

on them for incorporating ontological semantics into languages are used for four reasons. First, computer aided software engineering (CASE) tools and their model repositories are based on language meta-models. Using a language meta-model in assigning ontological semantics and expressing modelling rules as constraints on meta-model elements facilitates the incorporation of such rules into CASE tools. This in turn enables automatic or semi-automatic enforcement of real-world semantics for conceptual models.

Second, object-oriented languages, especially UML, are evolving languages. The evolution of such languages should be informed by theory. This work is aimed at providing such a theoretical basis. The language meta-model forms the ideal foundation on which to carry out the evoluation of a language, as it is well defined and unambiguous. On the other hand, graphical notations by themselves tend to be less well defined and more ambiguous. Hence, we employ the UML meta-model for purposes of informing the language evolution.

We suggest that ontological semantics should be incorporated into the language standard as long as this is not detrimental to the use of the language for IS design, which remains the main purpose of UML. Thus, the formal rules and constraints must not hinder the application of UML for IS design. Furthermore, the essence of the language must remain intact, i.e. object-oriented languages must retain support for e.g. encapsulation, classification, identification, generalization, message passing and other major concepts of the object-oriented paradigm.

Third, language meta-models are familiar to most users of the language as they define the use of a language. Thus, ontological semantics, when expressed in terms of the meta-model and constraints thereon, are immediately accessible to both the research and practitioner communities and can be put to immediate use.

Fourth, UML is a modelling technique incorporating many different diagrams or perspectives, which are unified by a common underlying meta-model. By mapping UML constructs into a coherent ontology by means of this meta-model, we can generate inter-diagram integrity rules to guide the construction of well integrated conceptual models.

Context This thesis will examine object-oriented design languages, and UML in particular, from the perspective of their suitability for generating conceptual models. The rules and ontological semantics derived serve to

enhance the quality of the models for the task of representing the business and organizational domain.

This must be put into the broader context of model and modelling quality. The frameworks by (Krogstie *et al.*, 1995; Lindland *et al.*, 1994), (Moody and Shanks, 1994, 1998; Moody, 1998) and by (Becker and Schütte, 1995, 1996; Rosemann and Schütte, 1997; Schütte, 1998; Schütte and Rotthowe, 1998; Rosemann, 1995; Schütte, 1999) provide some criteria for model and modelling quality, the adequacy of the language for the modelling task (Schütte, 1999) is but one of them.

Other important factors include the modelling cost (economic efficiency in (Schütte, 1999)), pragmatic quality with respect to the interpreter, social quality with repsect to a group of interpreters (Krogstie *et al.*, 1995), flexibility, simplicity, implementability (Moody and Shanks, 1994), correctness, comparability, clarity and relevance Becker and Schütte (1996).

These important dimensions of model quality are outside the scope of this dissertation. For example, the ontological semantics and rules are not intended to enhance the comparability or simplicity of models. However, as quality criteria are generally not independent, there may well be side-effects of the rules on these quality criteria. Two empirical studies, a case study 10 and an experimental study 11 are conducted, the results of which may address some of these questions. However, neither of these empirical studies are specifically intended to address these wider quality dimensions. Specifically, trade-offs between model cost and benefits of modelling as well as the efficiency of the model and modelling process are outside the scope of this thesis.

The remainder of this thesis proceeds as follows. The next chapter (Chap. 2) further describes the proposed methodology. As part of this chapter, section 2.3 introduces the concepts of the chosen ontology. This is followed by a review of previous work in the area of ontological semantics, formalization of UML, and domain analysis (Chap. 3). Our analysis of UML is done in three main parts. We examine fundamental static structure constructs first (Chap. 4), followed by constructs related to change (Chap. 5) and finally constructs related to interaction (Chap. 6), thereby covering all relevant aspects of the language.

Following the main theoretical analysis, Chap. 7 examines the proposed rules to ensure the rules do not violate fundamental object-oriented principles. This is followed in Chap. 8 by a generalization of the results to other

object-oriented languages. Chapter 9 provides two examples showing the effects which the proposed rules can have on conceptual models.

The second main part of this thesis is the empirical corroboration of the theoretical results, by a case study (Chap. 10) and an experimental study (Chap. 11). They serve to show the practical applicability and the specific benefits of the theoretical results. This thesis closes by pointing out the contributions to the knowledge of the field (Chap. 12) and future potential for extensions (Chap. 13) of this research.

Chapter 2

Methodology

It is clear to the software designer or programmer what any particular object-oriented language construct means in terms of the programming statements and code that ultimately results; these constructs possess implementation related semantics. However, it is much less clear to the business analyst what any particular object-oriented language construct means in terms of the business or organizational domain being analyzed; these constructs possess no real-world semantics.

In order to assign real-world semantics to a language, we must specify what exists in this world. Ontology is "that branch of philosophy which deals with the order and structure of reality in the broadest sense possible" (Angeles, 1981). A specific ontology makes assumptions about what exists and how things behave.

In this research, we propose to examine the usability of UML as a language for describing the real world by mapping its constructs to a set of real-world concepts, that is, to an *ontology*. This mapping will provide real-world semantics to UML constructs originally introduced to model IS elements. Our theoretical analysis rests on the following foundations:

1. Ontology : A philosophical commitment to existence
2. The BWW-Ontology: A specific set of ontological concepts and assumptions.
3. Ontological evaluation and assignment of real-world semantics.
4. Transfer of ontological assumptions and derivation of modelling rules.

5. Use of the UML meta-model to formally describe the derived rules.

The following subsections describe each of these foundations in more detail.

2.1 Ontological Foundations

Research into knowledge engineering, knowledge management and conceptual and domain modelling deals with representing the reality of the business or organization and has used *ontologies* for a number years. However, two different understandings of the word 'ontology' have evolved.

Research in conceptual modelling uses the term 'ontology' in its original philosophical sense, understood as meta-physics or the philosophy of existence (Angeles, 1981). Here, an ontology is a fundamental philosophical position, it is a commitment to the belief in the existence of certain entities in external reality. Research in conceptual modelling in IS has mainly drawn on philosophy and adopted a specific, well-developed, philosophical ontology (Bunge, 1977, 1979), although other ontologies (e.g. Chisholm, 1996) have been suggested as a basis for research. Once an ontological position has been adopted, it can only be cast in doubt by a gross inability to explain and predict observed phenomena (Kuhn, 1996). With the argument that knowledge representation languages should closely reflect external reality, ontology in this tradition has been used for analysis and evaluation of modelling languages (Evermann and Wand, 2001b,a; Gemino, 1999; Green and Rosemann, 2000; Opdahl and Henderson-Sellers, 2001, 2002; Parsons and Wand, 1991; Wand and Weber, 1989, 1993; Wand *et al.*, 1999). Empirical results are taken to confirm the commitment to this ontological foundation to be sensible, reasonable and adequate (Bodart *et al.*, 2001; Cockroft and Rowles, 2003; Gemino, 1999). Other research based on this foundation examines reference models (Fettke and Loos, 2003), provides a meta-model for the ontology (Rosemann and Green, 2002) and examines issues of data quality (Wand and Wang, 1995).

A specific ontology is a set of asssumptions about what exists in reality. Adopting an ontology is a fundamental philosphical choice that is necessarily prior to any science. As such, it cannot be justified or debated a-priori. As any philosophy, it is the framework that enables one to carry out science and research (Kuhn, 1996) and can only be assessed based on the results of that research. It should be noted that *an ontology is not a language*,

though it has to be expressed using a language. An ontology is a set of a-priori assumptions and commitments about the existence of entities that a language does not make. Ontology is universal. There exists only on world. The world, or an ontology, can be described by many different languages and using many different interpretations, but there exists only one world.

This philosphical understanding of ontology is distinct from the understanding of ontology in artificial intelligence (AI), knowledge engineering (KE) and computer science research, e.g. (Uschold and Gruninger, 1996; Noy and Hafner, 1997). In the AI tradition, the term ontology has come to signify a language or dictionary, a set of constructs to describe specific domains (e.g. medical, legal, manufacturing, etc.). Here, an ontology does not imply a firm commitment to the existence of a particular set of entities in reality, the connection to the real world have been severed: "Most of AI chose not to consider the work of the much older overlapping field of philosophical ontology, preferring instead to use the term 'ontology' as an exotic name for what they'd been doing all along in knowledge engineering ... It became correspondingly more remote from anything which might stand in a direct relation to existence or reality." (Smith and Welty, 2001, p. v). As such, AI-related ontology research does not make existence claims on a fundamental philosphical level.

Ontologies are understood as dictionaries, taxonomies, categorization schemata or modelling languages. AI research constructs its ontologies as needed. They are, after all, collections of words, a vocabulary or lexicon without commitment to any real, metaphysical existence in the world. The modeller or knowledge engineer is free to design or engineer ontologies (Gruninger and Lee, 2002; Holsaple and Joshi, 2002) to fit a particular problem or problem domain. Ontology is interpreted to signify a language that is specific to an arbitrarily broad or narrow set of users and an arbitrarily broad or narrow domain. Just like different languages may be employed to describe reality, in this research tradition, different ontologies may be employed to describe the domain knowledge. Consequently, in AI research, ontologies can be changed, adapted and customized to fit a specific purpose or domain (Gruninger and Lee, 2002). Hence, there is a need to evaluate ontologies (Guarino and Welty, 2002) to ensure their suitability for a particular purpose.

A return to philosophical ontology has been argued for e.g. by Guarino and Welty (2002, p. 61): "The computer science use of the term 'ontology' ... is taken as nearly synonymous with knowledge engineering in AI, con-

ceptual modeling in databases, and domain modeling in OO design. We believe it is important ... to maintain that 'ontology' is not simply a new word for something computer scientists have been doing for 20–30 years; ontology is hundreds, if not thousands, of years old, and there are many lessons learned in those centuries that we may borrow from philosophy along with the terms".

In the final instance, both interpretations of the word 'ontology' suggest that it is a set of concepts which can be used to describe what exists in the world. Work in the philosophical tradition treats the ontology as objectively given, while work in the AI and KE tradition treats ontology as constructed. This distinction is important, but has no bearing on the methodology that is described in the subsequent sections.

It is perfectly valid to employ the present methodology with an understanding of ontology and a specific set of concepts taken from AI related research. However, this would weaken the ties to external reality and would lessen the philosophical foundations of the work. It would furthermore be unclear whether an AI-style ontology should be substituted on the ontology side of the present research, as it is argued to be an ontology, or on the language side, as it is used as a language. When interpreting and examining AI-style ontologies as languages, the present methodology can be valuable for comparing the expressiveness of languages and deriving language translation rules. Thus, in the final analysis, besides the difference of whether the world (the ontology) is given (philophical ontology) or constructed (AI ontology), the distinction may simply be one of terminology.

However, this research takes up the call for a return to philsophical ontology and, in contrast to the AI research tradition, ontology is considered as a metaphysical philosophical commitment. While an ontology can be described by diffferent languages, some more suitable than others, the position taken in this thesis is that there exists only one ontology, one real world. This real world is objectively given and exists independently, it is not chosen, engineered or constructed by the modeller.

2.2 The BWW-Ontology

The specific ontology chosen for our purposes is based on Bunge's work (Bunge, 1977, 1979) as applied in a number of studies related to modelling in IS (e.g Wand and Weber, 1989, 1990, 1993; Wand *et al.*, 1999). We will

refer to this ontology as the BWW-ontology. Although other ontologies have been proposed as the basis for IS development [1] we choose Bunge's ontology as the basis for our work for a number of pragmatic reasons. As argued in the previous section, there can be no a-priori theoretical reasons for the choice of one ontology over another. In the end, the experience, observations and experimental results of science based on an ontology justifies the choice of that ontology. In this case, the successful application of the BWW-ontology and useful results from that research[2] justify our adoption of it.

- It is rooted in the ontological work done over a long period in the past: "Our work is in line with an old and noble if maligned tradition: that of pre-Socratic philosophers, Aristotle, Thomas Aquinas, Descartes, ... Peierce, Russell, and Whitehead" (Bunge, 1977).
- It is well formalized in terms of set theory and an axiomatic system.
- It has not been developed specifically for use in information systems analysis and design, but instead based on "the ontological presuppositions of contemporary scientific research, topped with new hypothesis compatible with the science of the day" (Bunge, 1977).
- It has been successfully adapted to information systems modelling and shown to provide a good benchmark for the evaluation of modelling languages and methods (Dussart *et al.*, 2002; Fettke and Loos, 2003; Green and Rosemann, 2000; Opdahl and Sindre, 1993; Opdahl and Henderson-Sellers, 1999; Opdahl *et al.*, 1999; Opdahl and Henderson-Sellers, 2001, 2002; Parsons and Wand, 1997; Wand and Weber, 1989, 1993; Wand *et al.*, 1999, e.g.).
- It has been used to suggest an ontological meaning to object concepts (Wand, 1989).
- It has been empirically shown to lead to useful outcomes by Bodart and Weber (1996); Bodart *et al.* (2001); Cockroft and Rowles (2003); Gemino (1999); Weber and Zhang (1996).

[1] Milton and Kazmierczak (1999) have used Chisholms ontology (Chisholm, 1996) as the basis of their investigation. Opdahl and Sindre (1993) suggest a number of concepts they argue are fundamental. However, the latter work is based on and limited to the semantics of data flow diagrams.

[2] See also Chap. 3.

These reasons are pragmatic in nature but provide a level of support, especially empirical support, that goes beyond that of other ontologies, and ontologies developed in the artificial intelligence and knowledge engineering research traditions. Thus, the adoption of the BWW-ontology as a true description of externally given and objectively perceived reality is justified.

2.3 Concepts of the BWW-Ontology

This subsection introduces the relevant ontological concepts, based on Bunge's work (Bunge, 1977, 1979), summarized in table 2.1.

The world is made up of substantial *things* that exist physically in the world. Therefore, entities[3] such as "addresses" and "jobs" are not things. Things can combine to form a *composite thing*. Composite things can be decomposed into parts that are in turn things. There exist basic things that cannot be decomposed (Bunge, 1977, Def. 1.1). Moreover, things cannot be created or destroyed, merely combined or broken up (Bunge, 1977, pp. 34f).

A thing possesses (substantial) *properties*. Properties in general are those possessed by a set of things, e.g. "color", "speed", "salary", etc. An individual property is one is representable as the value of a property in general, such as "blue in color", "speed of 100mph" or "salary of $2000" (Bunge, 1977, p. 63).

Properties can be either intrinsic or mutual. *Intrinsic properties* are ones that a thing possesses by itself, e.g. "color", whereas *mutual properties* exist between two or more things, e.g. "distance" (Bunge, 1977, p. 65). The co-domain of a property is some set. Hence, properties may be multi-valued if that set is a powerset. As an example, the salary of an employee of a company is a mutual property with a co-domain of a set of values specifying e.g. base pay, overtime pay and Sunday pay. Note that every powerset also includes the empty set. Moreover, no two things have exactly the same properties. Thus, properties can be used to identify things (Bunge, 1977,

[3]In this thesis everything that exists in the world will be called an entity. This is to avoid confusion with the notation of individuals and things in the ontology on the one hand and the notion of objects in UML on the other hand. When any of the latter are mentioned, the context is assumed to be the ontology or UML respectively. In cases where it is not clear from the context whether the ontology or the UML is referred to, the item will be prefixed, e.g. BWW-kind or UML-class. For the same reason a property of an entity shall be referred to as a feature as the terms 'property' and 'attribute' have a very specific meaning in either or both the UML and the ontology.

Ontological Concept	**Explanation**
Thing	Fundamental concept, the world consists of things.
Property	Things have properties.
Intrinsic Property	Property of one thing.
Mutual Property	Property of two or more things.
Law	Restriction on or relation of properties.
Composition	Things can be composed to form composite things.
Emergent Property	Property of a composite thing not possessed by parts.
Functional Schema	Set of state functions describing things.
State	Defined by values of state functions of schema.
Natural Kind	Set of things adhering to same laws.
Event	Pair of initial and final states.
Lawful transformation	Path in state space.
Process	Ordered set of events involving a single thing.
Interaction	State history is function of another thing.
System	Composite of interacting things.

Table 2.1: Concepts of the BWW-Ontology

Post. 2.5).

A *law* is any restriction on the property values of a *single* thing (Bunge, 1977, Def. 3.10). Specifically, law statements cannot relate the properties of different things. Law statements may be specified in different forms. Common forms of law statements relate the value of one property to those of other properties. Every substantial property must be lawfully related to some other (Bunge, 1977, Post. 2.7). Hence, there can be no properties which are independent (in the sense that they can change independently) of all others. Moreover, this also implies that a property must occur in at least one law statement (Bunge, 1977, Crit. 2.1).

Properties of composites may be either resultant (hereditary) or *emergent* (gestalt) properties (Bunge, 1977, Def. 2.16). Resultant properties are properties of at least one part of a composite whereas emergent properties are not possessed by any of the parts of a composite. Emergent properties can be explained in terms of or derived from properties of parts but are not reducible to them. Hence, emergent properties cannot be attributed to any of the parts by themselves. Some substantial properties of all composites are emergent properties. Hence, every composite must possess at least one emergent property not inherited from any of the parts (Bunge, 1977, Post. 2.9). For example, a computer composed of memory and processor possesses processing power, not possessed by any individual component.

Any thing can be characterized by a set of state functions. These functions correspond to properties of the thing (Bunge, 1977, Def. 3.9). They are usually functions of time indicating the value of the properties of a thing at a particular point in time (although other frames of reference are possible). Such a set of state functions is called a *functional schema* or *model*. Any thing can be described by more than one such schema (Bunge, 1977, Def. 3.6, Post. 3.4). For example a person may be described by functions indicating height and weight for one purpose, or described by location and organizational unit for another purpose. The *state* of a thing is defined as the set of values of all state functions (given a particular model) (Bunge, 1977, Def. 3.9). Specifically, states cannot be defined using a subset of state functions of a given model.

The *lawful state space* is defined by constraining the co-domains of the state functions to those values consistent with the laws that the thing adheres to (Bunge, 1977, Def. 3.11). A thing is always in a lawful state, i.e. a state within the lawful state space (even though that state may not always be desirable by an observer).

A set of things is called a *natural kind* iff there exists a set of laws that all the things in the set adhere to. Hence, for the definition of a natural kind the properties of things are irrelevant (Bunge, 1977, Def. 3.21). It is important to note that natural kinds are defined over an existing set of things. In this sense, the thing is the primary concept, not the natural kind. From this follows that there can be no natural kind without members. Since laws determine behaviour, a natural kind is the set of things that exhibit like behaviour.

Change may be quantitative, in which case the values of one or more properties is changed, or it may be qualitative (also called deep change), in which case properties are acquired or lost. The acquisition or loss of behaviour is generally concurrent with loss or acquisition of properties that changge in that behaviour. Change always involves the change of state of some thing. Since all things are changeable, it follows that the every lawful state space (recall there may exist different models) for a thing contains at least two distinct states (Bunge, 1977, Def. 5.1, Cor. 5.1). Then, change is defined as follows:

> *"A thing undergoes a* qualitative change *iff [the lawful state space] equals the union of at least two subspaces, each of which is spanned by a different projection of [the set of state functions]. Otherwise (i.e. if none of the components can be ignored during any stretch of the process), the thing undergoes only a* qualitative change*."*(Bunge, 1977, Def. 5.3)

This means that the quantitative changes in the two subspaces (for qualitative change) are independent of each other.

Rather than assigning things a new name on every change of a property, Bunge advocates keeping the name of a thing until it changes it's natural kind (*principle of nominal invariance*):

> *"A thing, if named, shall keep its name throughout its history as long as the latter does not include changes in natural kind – changes which call for changes in name."*(Bunge, 1977, Princ. 5.1)

Change may be represented either through a description of events as ordered pairs of states or through a description of processes. An *event* is any pair of states that are part of a state space of some thing (Bunge,

1977, Def. 5.4). Note that events are defined as pairs of states of *the same* state space. Hence, qualitative change cannot be described in this form. Moreover, an event is defined for the state space of a single thing. Hence, an event cannot involve two or more things. It is of course possible that a change in one thing leads to a change in another, but this is interaction consisting of two distinct events in the two things. Events can compose to form a complex event if the final state of the first event is the initial state of the second event. Such a complex event is called a process. Many different processes can have the same initial and final state (Bunge, 1977, Def. 5.6).

If the state space is non-denumerable, e.g. because one or more state functions are defined with a co-domain of real values, there exist infinitely many intermediate states between an initial and a final state. In this case change is represented by a triple (s_i, s_f, g) representing the initial state the final state and a function g which represents the path in the state space the thing traverses (Bunge, 1977, Def. 5.8, Princ. 5.3). The *lawful transformation* g is a function from the lawful state space into the lawful state space. Hence, it is compatible with the laws that a thing adheres to and can itself be thought of as a law or set of laws (*transition laws*).

A *process* is an ordered set of events that involve a single thing. The simplest type of process is serial change, i.e. a chain of events in a thing. In general, a process may not be a chain but can be "envisaged as a directed tree [...] in some event space" (Bunge, 1977, p. 243). Thus, processes may be diverging or converging series of events. Processes have a defined beginning and end (Bunge, 1977, Post. 5.8).

Interaction is defined through the state history of a thing: If the way attributes of one thing change depends on the presence of another, then the second is said to act on the first (Bunge, 1977, Def. 5.29). Things interact, if and only if each acts upon the other (Bunge, 1977, Def. 5.30) and every thing acts on, and is acted on by other things (Bunge, 1977, Post. 5.10). Finally, every thing undergoes some spontaneous changes and some externally induced ones (Bunge, 1977, Post. 5.11).

A link between two things is any relation between two things, e.g. "thing A is behind thing B", "thing A is older than thing B" whereas a connection or a coupling makes a difference to the things, i.e. two things are connected (coupled, linked, *bonded*) if at least one of them acts on the other (Bunge, 1979, p. 6). A *system* is defined as a composite whose parts are bonded, i.e. there exists interaction among all the parts (Bunge, 1979, Def. 1.1). Every system is acted on by its environment and acts on its environment (Bunge,

1979, Post. 1.1, 1.2). Systems are assembled from parts (i.e. the parts begin interacting) and every such assembly is accompanied by the emergence of some properties and loss of others (Bunge, 1979, Post. 1.4, 1.5).

Remarks There are a number of noteworthy things to mention. First, the BWW-ontology makes no claim to a mechanism of interaction. Interaction of things is defined in phenomenological terms and Bunge (1977) only postulates a criterion for recognizing when interaction has occurred. Note however, that all possible change must be in accordance with laws and change happens as a result of laws. If two properties are lawfully related and one changes, then the other may also change. Moreover, laws relate properties of one thing only. Hence for a thing A to act on a thing B as the result of laws, there must exist either a mutual property of A and B or an emergent property of the system composed of parts A and B. Properties of A and B must be lawfully related to the mutual property or the emergent property. Note that while the existence of such a property is a consequence of the ontological assumptions, it does not constitute a mechanism of interaction.

Second, interaction may give rise to properties, either emergent or mutual ones (e.g. a student enrolls at a university $\Rightarrow$ "tuition fee balance"). Hence, there exist some properties that must necessarily exist prior to interaction and some that may exist post interaction.

Generally, most interactions will give rise to some mutual properties. Moreover, since these mutual properties will not generally disappear, this interaction history may be represented by the mutual properties acquired by a set of things during the course of their interaction.

Since behaviour is governed by the laws that things adhere to, the same set of laws leads in principle to the same (potential) behaviour. Of course, different initial conditions may also play a role. Nonetheless, we suggest that since a natural kind is defined in terms of its laws, a natural kind may also be characterized as the set of things that exhibit in principle the same (potential) behaviour.

Third, while the BWW-ontology appears very materialistic and physical, it has enough descriptive power to give interpretation to what might be called 'conceptual constructs' that are not physically existent in the world. For example, a customer's order of a product from a supplier is not a thing, but rather an event, an interaction between the supplier and the customer. As such, an order history is a set of events.

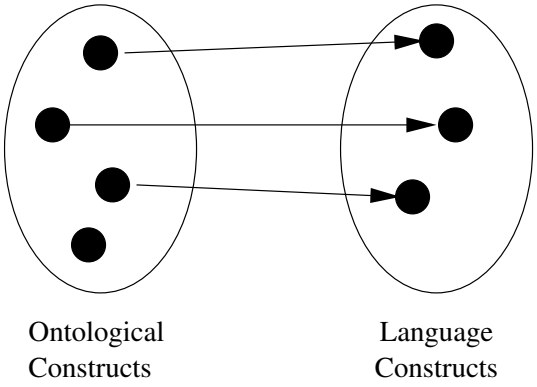


Figure 2.1: Construct deficit

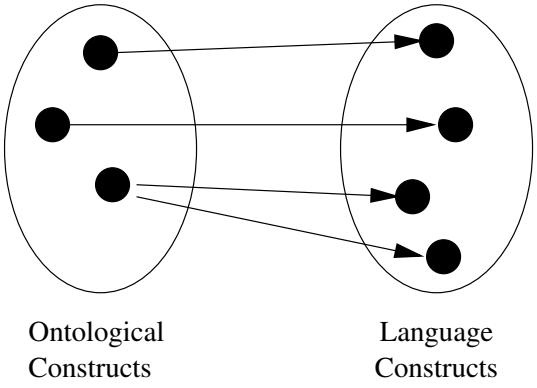


Figure 2.2: Construct redundancy

2.4 Assigning Real-World Semantics

Assigning ontological meaning to a language amounts to answering two questions (Wand and Weber, 1993):

1. How can an element of the real-world domain (ontological concept) be *represented* in the chosen language?

   To answer this question we propose a *representation mapping* from the set of ontological concepts into the set of language constructs which assigns each ontological concept a language construct with which to

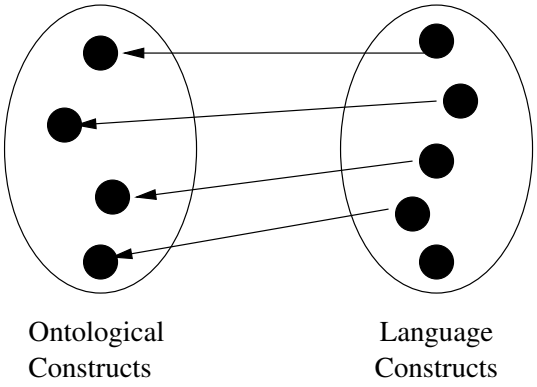


Figure 2.3: Construct excess

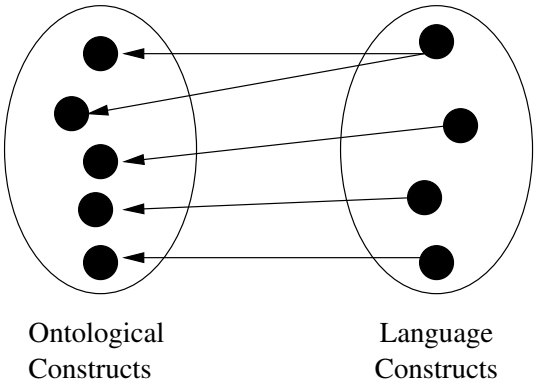


Figure 2.4: Construct overload

represent it.

Analysis of the representation mapping can identify *construct deficits* (Fig. 2.1), situations where the modelling language does not provide a construct to represent an ontologically relevant aspect of the real world. This may lead to incomplete models when the language is used for conceptual modelling. A second potential defect is *construct redundancy* (Fig. 2.2), where an ontological concept can conceivably be mapped to two different language constructs. This may lead to confusion as the modeller has no guidelines on which construct to use.

2. How can a construct of the language be *interpreted* in terms of the real-world domain (ontologically)?

   To answer this question we propose an *interpretation mapping* from the set of language constructs into the set of ontological concepts which assigns each language construct an ontological interpretation.

   Analysis of this mapping can identify language constructs that have no ontological interpretation (*construct excess*, Fig. 2.3) or have multiple ontological interpretations (*construct overload*, Fig. 2.4). Use of a construct without ontological interpretation may lead to an ontologically meaningless model. Construct overload can lead to ambiguous models as it is unclear which interpretation to chose. This can lead to misunderstandings and misinterpretations during the analysis and design process and result in a faulty information system.

Together, these mappings assign real-world, ontological semantics to a modelling language. A mapping provides maximum ontological clarity (a minimal number of defects) if it is bijective (Wand and Weber, 1993), i.e. a one-to-one mapping that maps all elements.

Our analysis will propose representation and interpretation mappings and analyzes the mapping for defects. Specifically, construct deficits can be alleviated by suggesting new language constructs. Other defects do not require additions to the set of language constructs, but can be solved by providing appropriate modelling rules, described presently.

2.5 Derivation of Modelling Rules

There are two reasons to introduce modelling rules. First, relationships between ontological concepts should be reflected in the language. Second, the

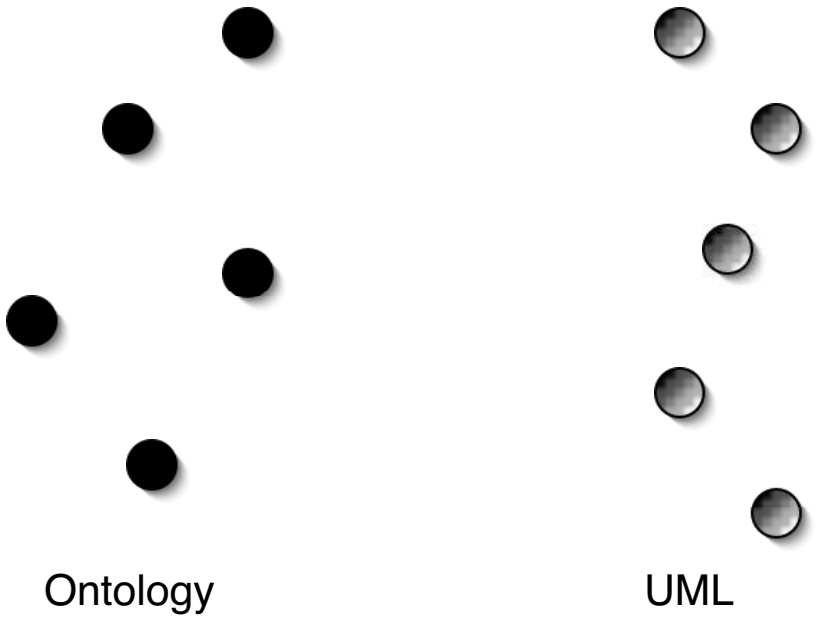


Figure 2.5: Step 1: Identify ontological concepts and language constructs

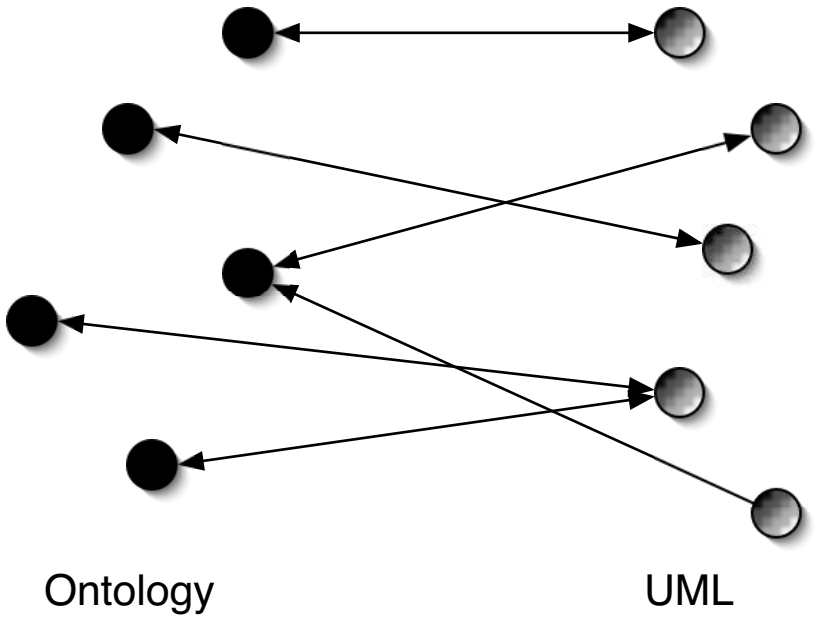


Figure 2.6: Step 2: Map ontological concepts to language constructs

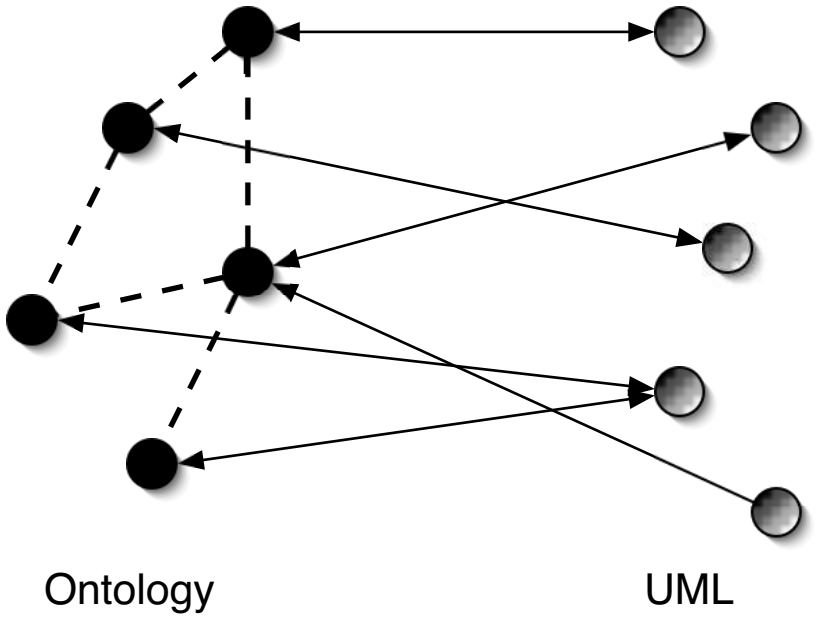


Figure 2.7: Step 3: Identify ontological assumptions, relationships and constraints between concepts

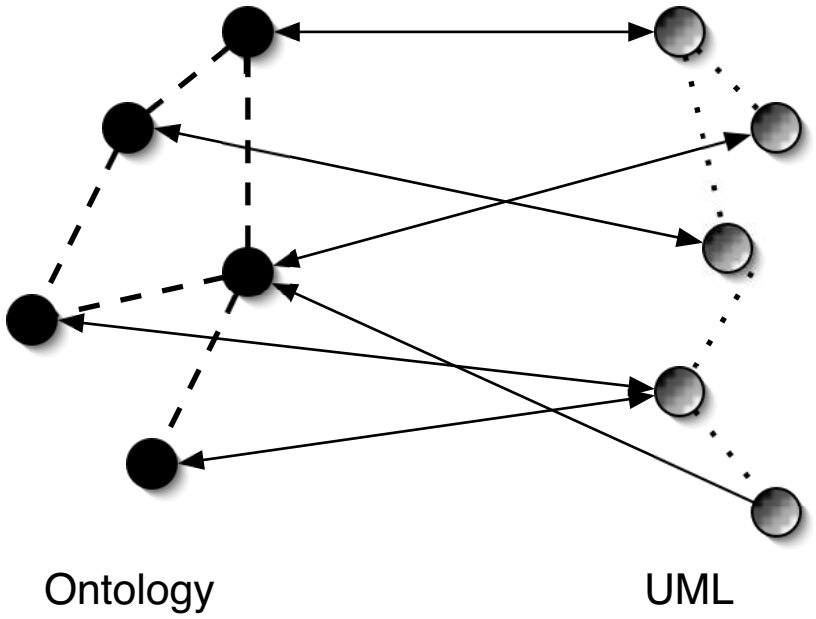


Figure 2.8: Step 4: Transfer ontological relationships by virtue of mappings, thereby deriving modelling rules

representation and interpretation mappings should exhibit minimal ontological defects.

Transfer of Assumptions Once the ontological semantics are assigned and language constructs are mapped to ontological concepts, the mappings can be used to transfer ontological assumptions to the language. An ontology may suggest that certain situations are possible in the real world while others are not. By virtue of the mappings, some combinations of language elements may therefore describe possible real world situations while others may describe impossible ones. Thus, if there are rules or constraints that relate ontological concepts, then by virtue of the mapping, these same rules or constraints must also hold between the mapped language constructs, in order to allow only models of possible real-world situations. Hence, the ontological mapping can lead to modelling rules on *how* to use the language for conceptual modelling.

Figures 2.5 through 2.8 show this process. First, the relevant ontological concepts and language constructs are identified (Fig. 2.5). In a second step, both representation and interpretation mappings are proposed (Fig. 2.6, Sec. 2.4). Third, relationships between ontological concepts are analysed (Fig. 2.7). Fourth and last, these relationships are then transferred to those language constructs that are mapped to the ontological concepts (Fig. 2.8), thereby generating modelling rules. These rules constrain the use of the language in such a way as to allow only the modelling of possible real-world situations.

For languages with a well-formalized syntax such as UML, the proper usage of language elements can be enforced by proposing constraints on the meta-model elements. The constraints restrict the set of possible models to allow only the modelling of possible real world situations. These meta-model constraints formalize real-world modelling rules in the syntax of a language; they govern the way in which language constructs may be combined to form ontologically valid models.

Ensuring Ontological Clarity A mapping provides maximum ontological clarity if it is bijective, i.e. a one-to-one mapping that maps all elements. Mappings which are not bijective exhibit undesirable *ontological defects* (Wand and Weber, 1993).

To achieve a bijective mapping, it may be necessary to map language

constructs to ontological concepts only if they appear in a particular context: Suppose that a language provides a construct which, depending on the context in which it is used, expresses two different real-world situations. Ontological clarity can be achieved by proposing appropriate modelling rules which map that construct, when used *in the first context* to the one ontological concept and map it, when *in the second context*, to the other ontological concept. Such rules can be enforced by placing constraints on the meta-model.

There are two important things to note about this method. First, while the derived rules may be appropriate for conceptual modelling, they might not be obvious or applicable when the language is used for IS design purposes only. It is not the purpose of this work to show modelling rules for IS design. Instead, we intend to *extend* the use of design languages into real-world organizational and business modelling. Second, such rules do not necessarily guide us in how to perceive the world. Thus, we might suggest rules on how to model objects and classes, but not on how to identify them.

2.6 Use of Meta-Models

A meta-model is a description of the language elements and the language syntax in the same or another language. The UML meta-model is specified in UML itself, with the help of the object constraint language OCL, which is officially a part of UML. OCL is used in the specification of UML to restrict the possible combinations of model elements. The UML specification OMG (2001) makes extensive use of this mechanism. As the UML meta-model itself is a UML model, this research also makes use of OCL to formally describe constraints on UML model elements.

The lack of a meta-model and a widely known language for the BWW-ontology has been partially addressed by Rosemann and Green (2000, 2002) who propose a model based on entity relationship (ER) diagrams. This meta model could serve as the basis for comparing modelling languages or methodologies (Davies *et al.*, 2003; Rosemann and Green, 2002; Rosemann and zur Mühlen, 1998). However, their model omits central elements of the BWW ontology such as states and state transitions. Also, the UML meta-model is formalized in UML itself, not ER diagrams. These two reasons prohibit the use of a formal meta-model based approach, e.g. schema matching (Batini and Lenzerini, 1986; Rahm and Bernstein, 2001), for the time being. This

last point is also recognized as a hindrance by Davies *et al.* (2003) who encourage the use of meta-models for comparing ontologies and languages and suggest that such kinds of analyses can benefit from meta-models by being forced to be comprehensive.

Any proposed changes to the meta-model stem from three sources:

- Construct deficits:

  Problems of missing constructs to express ontological concepts can be solved by introducing new language elements. These new elements are additions to the meta-model.

- Lack of ontological clarity:

  When a naive mapping, a mapping that does not consider relationships among language constructs or among ontological concepts, leads to ontological defects such a construct overload, modelling rules can be proposed to limit the mapping to certain contexts in which the language construct is used. These limitations may be expressed by OCL constraints on UML meta-model elements.

- Transfer of ontological assumptions:

  Relationships and constraints that exist between ontological concepts can be transferred to the language by virtue of an ontologically clear mapping. If a relationship r_o exists between two ontological concepts O_1 and O_2 which are mapped to language constructs L_1 and L_2 respectively, we can propose a relationship r_l in the language between constructs L_1 and L_2.

  Such relationships are constraints on possible real-world situations. Language elements should be usable in only such a way as to allow models of possible real-world situations and deny modelling impossible real-world situations. In simple cases such relationships or constraints may be expressed by adding a relationship or modifying an existing relationship. More complex cases may additionally involve OCL constraints.

The three sources of changes can lead to two types of changes:

- Additions or changes to the UML meta model elements:

Additions or changes to the UML meta-model elements can take a number of forms, e.g. changes in association multiplicities, addition of associations between formerly unrelated meta-model elements, or additions of classes of language elements[4].

We attempt to restrict changes to the UML meta-model to a minimum and instead prefer to employ OCL. Our aim is not to propose a meta-model of the BWW-ontology in UML, nor do we want to propose changes to the UML meta-model which would lead to an exact match with the BWW-concepts and assumptions. UML should remain an object-oriented language whose primary use is for IS design. We aim to find an ontological interpretation which requires the least reinterpretation and preserves the original language elements and their valid combinations as much as possible. We wish to preserve as much as possible the constructs and syntax necessary to support IS design and all object-oriented principles such as encapsulation, classification, message passing, etc.

As a consequence, we refrain from changing aspects such as the use of composition or aggregation instead of ordinary associations. An example of this is the definition of an "Instance" in the meta-model as a composite of zero or more "AttributeLinks" (values of attributes). While this is correct for a description of a software design, instances in the real world are of course described by, not composed of, one or more attribute values. Hence, this should be modelled by an ordinary association. There exist a multitude of similar cases in the UML meta-model but we refrain from suggesting changes to these. Instead, we rely on the reader of a model to interpret the semantics of these associations with respect to the real world.

- Specification of constraints in OCL:

  Often the ontological assumptions can be expressed using the elements of the existing meta-model or the suggested alterations to the meta-model. In these cases changes to the meta-model elements are not required and instead we rely on OCL to specify our proposed rules formally.

[4]All additions to the meta-model and OCL expressions are based on version 1.4 of the UML meta-model OMG (2001). A full discussion of the meta-model is beyond the scope of this paper, the reader is assumed to be familiar with the model. Hence, additions to the meta-model are often suggested without depicting the larger context in which these are embedded.

The changes to the UML meta-model are syntactic in nature and by themselves do not affect the semantics of UML vis-a-vis the real world. These semantics are determined by the ontological mappings that we assign, not by the meta-model.

2.7 Scope of Mappings

Chapter 1 outlined our motivation for incorporating ontological semantics into language meta-models. Besides facilitating CASE tool support, meta-model level work allows inter-diagram considerations and modelling rules. UML as a graphical language comprises nine distinct diagrams: use case diagram, class and object diagrams, sequence and collaboration diagrams, statechart and activity diagrams, component and deployment diagrams. Not all of these are necessary for every purpose. In fact, the last two in this list are considered implementation diagrams, concerned with the physical packaging and distribution of software to hardware components. However, the UML meta-model provides the underlying integration of these diagrams by providing relationships between language constructs used in different diagrams. Thus, a meta-model analysis allows a *comprehensive* and integrated analysis of all UML diagrams.

Such a comprehensive approach is important. Any assignment of semantics to language constructs should not be done individually for each construct. Instead, it must be done before the background of the whole language as an interconnected set of constructs, not as a set of independent symbols. Furthermore, the interpretation and representation mappings should attempt to preserve the relationships among language constructs, whether they are formal syntactic constraints or only exhibited in common usage.

As an example, consider the UML-construct "object". UML suggests that each object may have a UML-construct "state" associated with it. Any mapping of "object" and "state" should attempt to preserve this relationship between the concepts to which "object" and "state" are mapped in the ontology. If necessary, different mapping must be suggested for "objects" and "states" in different contexts.

The alternative, individual mapping of constructs and disregarding any relationships between them, may lead to simpler representation and interpretation mappings. However, these may turn out to contradict ontological assumptions. For example, mapping action states to ontological states will

contradict the ontological assumptions that state transitions occur within a single entity. When that is the case, either the ontological assumptions must be dropped, or the existing language meta-model must be discarded. To do the first is impossible, as the ontology is externally given, while the second suggestion may significantly alter the language.

Despite this argument for a comprehensive approach, this thesis does not consider the language constructs used in the two implementation diagrams, component and deployment diagrams, nor those in use case diagrams. These three diagrams explicitly contain the information system or parts thereof as model elements. As this thesis is concerned with conceptual modelling of the business and organization, *not* the software, these diagrams are not relevant to the purpose of this thesis.

Furthermore, these three diagrams model the information system in the context of entities such as organizational actors, hardware, etc. which are external to the IS, e.g. actors interacting with a use case, packages being distributed to hardware nodes. As the IS is a representation of the real word, these IS-external entities must consequently be representations of things external to the real world. This is clearly not possible. Hence for the purpose of assigning business semantics to UML for conceptual modelling, these diagrams are not relevant. The IS-external entities in these diagrams cannot be mapped to any ontological concept, as there exists no such concept that is external to the real world. Consequently, for the purpose of conceptual modelling, they are ontologically excessive.

Chapter 3

Prior Research

Research focussing on the analysis of domains for IS design has a long tradition in software engineering under the name of domain analysis. Similarly, research to formalize UML is not new and ontologies have been used for the analysis of IS modelling languages before. These three streams of research have had little contact with one another and different aims and goals. This chapter examines the relevant work and points out weaknesses and differences to the current work.

3.1 Domain Analysis

In the area of software engineering or software design, it has been recognized that, for effective reuse of software artifacts, an understanding of the application domain is necessary. Within that research area, the term domain analysis has come to signify the attempt to understand the area of application of a software system. Domain analysis attempts to abstract from the specifics of a particular IS and attempts to model the features of the application domain in order to build artifacts that enable reuse.

Most of this research views domain analysis not necessarily as the analysis of the real-world but rather as the first step in the development of a reusable and expandable software architecture or framework (Arango, 1989; Morandin *et al.*, 1998; Philippow and Riebisch, 2001; Jayase *et al.*, 2001). Thus, a domain model is not necessarily a faithful representation of the real world. Domain analysis then is the identification and documentation of com-

monalities and differences that can occur across different implementations of a domain framework or domain architecture.

Object-oriented techniques have been proposed for domain analysis as those techniques became increasingly popular with software engineering (Gomaa, 1992). The focus of this early use of object-oriented techniques has been the identification of variability in a domain, following the promise of more flexible software design by OO proponents. Therefore, domain analysis using object-oriented techniques is able to explicitly support reuse of software components (Cohen and Northrop, 1998) and the domain models include representation not only of the physical world, but also elements of data abstraction, control and algorithms (Gomaa, 1992).

This focus on software reuse is also evident in (Galfione *et al.*, 2000): "[domain analysis is] a methodology to support the identification, collection and organization or the artefacts used in software development". Only to a secondary degree is domain modelling concerned with real-world models. Instead, it should specifically take into account IS considerations such as reuse and software configuration issues (Cohen and Northrop, 1998).

The general process of object-oriented domain analysis begins with domain exploration and modelling followed by specification of systems and architectures which include 'hot spots', points of variability that must be configured to yield concrete models (Morandin *et al.*, 1998). With the increasing calls for object-oriented techniques for domain analysis (Cohen and Northrop, 1998; Gomaa, 1992), the use of UML has also been suggested. However, UML cannot express variability required for domain models to adapt to various similar domains and needs to be expanded to include constructs for instantiating concrete models (Morisio *et al.*, 2000; Philippow and Riebisch, 2001). In this sense, domain analysis is closely related to reference models, which have also been examined ontologically. However, a different methodology than the one proposed here must be used for this (Fettke and Loos, 2003).

Domain analysis embraces object-oriented techniques for the purpose of constructing models for software reuse. This contrasts with our understanding of conceptual modelling which specifically excludes any software considerations. Consequently, proposals to extend UML for domain analysis focus on expressing variability, not on real-world semantics. Domain analysis does not distinguish between the real world and the real world as viewed for the purposes of IS design. Moreover, real-world semantics for UML are implicitly assumed and taken for granted. In contrast, this re-

search does not accept the suitability of object-oriented languages a-priori. Instead the primary objective of this research is to assign semantics to ensure this suitability. In this way, the results of this research can further the goals of domain analysis.

3.2 UML Semantics

Another related stream of research is the formalization of UML. This research has taken UML from a purely graphical language with few rules to a language based on a rigorous meta-model that governs the application of language constructs. An important result of the precise UML (pUML)[1] group and the drive for formalization has been the UML meta-model (Evans and Kent, 1999; Evans *et al.*, 1999a) which has been described in UML itself with the help of the object constraint language OCL. This meta-model has been adopted into the language standard (OMG, 2001) and is used as one of the foundations of this work.

This section briefly describes recent efforts at assigning not only formal syntax but also formal semantics to UML.

Breu *et al.* (1997, 1998b) motivate the formalization effort by arguing that a formal semantics provides advantages for modellers and tool developers. They recognize that UML models consist of different diagrams expressing various views or perspectives which must be integrated. Hence, UML constructs are mapped to various mathematical and logical models such as SYSLAB and Z (Breu *et al.*, 1998a; Bruel and France, 1999).

The pUML group works towards a semantics for UML by formalization in logic. This kind of semantics is concerned either with internal consistency (and is thus a coherentist view of semantics) or defines semantics of an IS as the input/output relationship or transformation rules (and is thus a procedural semantics). Evans *et al.* (1999b) propose a formalization of UML by specifying it in the formal language Z, arguing that Z better supports logical inferences and proof than the OCL language which is part of UML. Evans (1998); Evans and Clark (1998) show how UML class diagrams can be formalized in Z. Lano and Evans (1999) relate UML diagram elements to a first order set theoretic model called RAL (Real-time Action Logic) and demonstrate how this can lead to transformation rules that can take UML models from analysis to design.

[1] http://www.cs.york.ac.uk/puml

Various other works claim to assign formal semantics to UML. Lano and Bicarregui (1999) use formalization as a tool to derive transformation rules for UML models, Lilius and Paltor (1999) provide a formalization of UML state machines and Övergaard (1999); Övergaard and Palmkvist (1999); Knapp (1999) suggest a formal treatment of interaction, especially collaborations and use cases.

However, all of this research relates to internal consistency and the formalization of UML in logic and mathematics, not to its relationship to the real world. Specifically, these kinds of semantics are not correspondence semantics, i.e. the meaning of constructs is not determined by their relations to extra-linguistic elements, such as real-world elements. By not relating the language to extra-linguistic entities they fail to impart real-world meaning to UML constructs. Thus, none of these approaches make clear the meaning of UML constructs in the business or organizational domain. They answer the question of what a particular UML construct means e.g. in the Z language, but this simply moves the problem to the real-world semantics of Z.

3.3 Ontological Analysis

This section describes research on how ontology and specific ontologies have been used for the analysis of IS modelling languages. While the research discussed in the last section lacked this element, most of the work discussed in this section lacks the formal aspect that is common to research discussed in that section.

The use of ontology in IS research dates back to the work by Wand (1989) who introduces the main concepts of Bunge's ontology to the IS field and examines five principles of object-oriented concepts. He goes on to suggest ontological semantics for very general object-oriented concepts without examining a specific language. Early work (Wand and Weber, 1989) identifies implementation related languages and constructs that are distinct from representation related constructs. Only the latter are deemed suitable for representing real-world domains, while the former relate to software artifacts. Among such implementation related constructs are messages and message passing concepts (Parsons and Wand, 1991, 1997). This finding reflects the heritage of the object-oriented approach from software engineering (Opdahl and Henderson-Sellers, 2001).

Only later is the notion of ontological evaluation developed and applied

to data flow diagrams (Wand and Weber, 1993). Next to classification theory (e.g. Lakoff, 1987) and speech act theory (Austin, 1962; Searle, 1969; Auramaki *et al.*, 1988, e.g.), ontology is proposed as a third basis for evaluating models and modelling languages.

Parsons and Wand (1997) describe the applications of ontology in general, and the BWW ontology in particular, to the evaluation of systems modelling techniques and use it to the clarification of the notion of object in systems analysis. Specific focus on object-oriented techniques has led to a proposed modelling process (Wand and Woo, 1999) for object-oriented languages that extends from real-world conceptual modelling to high level IS design (Wand *et al.*, 2000). The modelling rules and process are generic and not specific to a particular language. Wand and Woo (1999) do not provide detailed syntactic rules. As such, the present research is complementary to their process.

Recent work has focussed on specific languages and even specific language constructs such as the relationship construct in entity relationship (ER) diagrams (Wand *et al.*, 1999). As object-oriented techniques gained popularity in IS design, various studies have examined the OPEN Modelling Language (OML). OML is an experimental variant of UML and, when analyzed using the BWW ontology, exhibits a number of overloaded, excessive or redundant language constructs as well as construct deficits (Opdahl and Henderson-Sellers, 1999; Opdahl *et al.*, 1999; Opdahl and Henderson-Sellers, 2001). However, no modelling rules have been proposed.

Ontological analysis has also been successfully applied to more traditional, non object-oriented languages. The ARIS language (Scheer, 1999) is a combination of four distinct and slightly overlapping languages. Ontological analysis indicates that even as a combination of languages, ARIS exhibits ontological constructs deficit and cannot express the BWW-ontology completely (Green and Rosemann, 2000).

On a more critical note, Rosemann and Green (1999) suggest that ontological analyses neglect the purpose of the model and characteristics of the user or modeller. Thus, they argue, the deficiencies identified through ontological analysis may not be relevant to practical modelling situations. They demonstrate this using examples of activity based costing and workflow modelling and suggest that the ontologies used should be either amended or restricted to fit the purpose and user. However, as argued above, an ontology is a most fundamental philosophical position and as such is not tied to a specific purpose. It should be departed from only if the results of

research based on that position obviously violate accepted truth. Moreover, some languages are not intended to completely express the BWW ontology (Opdahl and Henderson-Sellers, 2002; Parsons and Wand, 1991, 1997), and changes to the BWW-ontology to accomodate e.g. software specific constructs should not be considered.

Analysis of UML Two other research groups have used the BWW-ontology to analyze the UML language (Opdahl and Henderson-Sellers, 2002; Dussart *et al.*, 2002), the former being more comprehensive than the latter. Both works stop short of suggesting modelling rules to be incorporated into the meta-model, but instead analyze the language for ontological deficiencies. This chapter reviews the different choices those authors made in mapping ontological concepts to UML language elements. Since a full discussion of the research is beyond the scope of this thesis, the mappings of the most fundamental constructs and concepts will be discussed. The end of this section draws some general conclusions from this discussion.

Both (Opdahl and Henderson-Sellers, 2002) and (Dussart *et al.*, 2002) map UML-objects to BWW-things, but in order for UML-objects not to be overloaded, this interpretation mapping needs to be restricted to those objects which are substantial. Other UML-objects must be mapped to e.g. BWW-properties, and the modeller must be given a rule for this mapping. The representation mapping from BWW-things into objects is not problematic. Dussart *et al.* (2002) map a BWW-thing also to a swimlane, or more formally, a partition in UML. However, this leads to state transitions between two objects, which, when mapped back to BWW-things, should not be possible.

Opdahl and Henderson-Sellers (2002) as well as Dussart *et al.* (2002) map BWW-classes to UML-classes. However, this neglects the fact that a UML class is a specification, *not* a collection of things like a BWW-class. Collections of things are modelled as aggregates and composites in UML.

BWW-properties are mapped to UML attributes (intrinsic properties) and associations (mutual properties) by Opdahl and Henderson-Sellers (2002) while (Dussart *et al.*, 2002) make the unorthodox mapping of BWW-properties to UML-activities and UML-partitions. The first mapping has to contend with the fact that properties are mapped to two very different UML-elements, while the second mapping brings with it the problem that partitions are not features of UML-objects, so that the BWW-properties mapped to them are not actually features of UML-objects. This mapping also over-

loads the partition construct to represent both things and properties.

Both Opdahl and Henderson-Sellers (2002) and Dussart *et al.* (2002) map BWW-states to UML-states and both attempt to find a mapping for the BWW-state history of a thing, the collection of states through time of a thing. Dussart *et al.* (2002) map the history to the UML-shallow history construct, which does not fully capture the meaning of the BWW-history concept. Opdahl and Henderson-Sellers (2002) on the other hand map the BWW-history to a UML-object lifeline, which, while often shown in sequence diagrams, is not a formal UML element defined in the meta-model. While this captures some sense of a thing's history, it does not, or only indirectly by tracing messages to state transitions, reflect the series of states that a thing has gone through.

Interestingly, Dussart *et al.* (2002) map the conceivable state space of a thing to a UML-state machine and the lawful state space to the set of sub-states, recognizing that a state machine *contains* states just like a state space contains states. Opdahl and Henderson-Sellers (2002) find no counterpart in UML corresponding to the conceivable or lawful state spaces. The mapping to UML-state machines however requires that these are state machines which are associated with a classifier, i.e. those which specify the behaviour of instances or objects, *not* those which specify the behaviour of methods. This must be ensured by an appropriate modelling rule. While this is a step towards capturing the meaning of the conceivable state space, one would also need to include states and sub-states of all state machines which define behavioural features of UML-objects. With this extended reading, the mapping by Dussart *et al.* (2002) is a good interpretation mapping.

Opdahl and Henderson-Sellers (2002) map ontological stable states to UML-final states and vice versa. This neglects the fact that BWW-stable states are defined through interaction, a concept which should be mapped to UML-events or actions. Hence intermediate states may also be considered stable, if state transitions from such states are caused by external interaction.

Opdahl and Henderson-Sellers (2002) propose an interpretation mapping which maps most interaction related constructs such as transitions, guard conditions, actions, etc. to BWW-laws. On the other hand, a BWW-law is not mapped to the obvious equivalent of a UML-constraint. This mapping is problematic without clear guidelines to the modeller as it shows a lot of ambiguity with respect to the representation of BWW-laws in UML.

There are a number of important things to note. First, in Dussart *et al.* (2002) it is unclear whether the mappings given are interpretation or representation mapping or both. Also, their research is restricted to UML elements used mainly in activity charts and concerned with interaction. Both Opdahl and Henderson-Sellers (2002) and Dussart *et al.* (2002) suggest mappings that clearly show ontological deficiencies such as overloaded language constructs or construct redundancy. This is the result of trying to map the elements without restricting the applicability of the mappings to make the mappings less problematic. Their conclusions, that the languages are deficient, is a direct result of this. On the other hand, our research attempts to find rules by not only transferring ontological assumptions to the UML-language but also, and perhaps more importantly, suggesting ways in which to enhance the ontological clarity of the mapping and reduce the ontological deficiencies. For example, we restrict the mapping of UML-objects to BWW-things to substantial objects only, thus reducing ontological deficiencies and providing a clear modelling rule.

Second, there exist elements in the UML-meta-model which have no semantic role to play. These are not mapped to BWW-elements in the present research and no modelling rules need to be derived. Other UML-constructs such as sync states, focus of control, timing mark, etc. that are mapped by Opdahl and Henderson-Sellers (2002) are not mapped in this thesis due to the following reasons. First, we believe that these are very UML specific constructs that have no general object-oriented counterpart or meaning. Second, these constructs are specifically related to IS implementation. Third, these constructs, if they were mapped, would result in ontological deficiencies in the mapping, such as construct overload. In contrast to Opdahl and Henderson-Sellers (2002) we have therefore decided not to map such UML-constructs.

To summarize, previous work has concentrated on ontological evaluation of languages. In contrast, our present aim is to employ ontology in a more prescriptive way and assign ontological semantics to a language. The present work builds on earlier work (Evermann and Wand, 2001a,b) to use the BWW ontology not only as the basis to determine ontological deficiencies, but also in a constructive way: Through the interpretation or representation mapping we can assign real-world meaning (ontological semantics) to language elements. Hence, this tells us not only *whether* to use such languages to model the real world, but also *how* to use them. This work also builds on the formal syntax and formal semantics of the UML meta-model and the object constraint language, attempting to bridge the two diverse streams

of research. It is the first study to employ formal descriptions of the rules resulting from an ontological analysis.

Chapter 4

Static Structure

Our world consists of a static structure of things with their properties, changes in things and interactions of things. Hence, this is the order of the following discussion. Each of the following chapters first employs a representation mapping, followed by an interpretation mapping. From these mappings a number of modelling rules follow as consequences. These are first stated informally and then formalized using the UML meta-model and OCL. As stated in Sec. 2.5, the derived modelling rules are intended for conceptual modelling of real-world domains. They are not applicable for use of UML in software design and software modelling.

4.1 Representation Mapping

The representation mapping is intended to find a first UML representation of the basic ontological notions. The basic building blocks of the world are things, which possess intrinsic and mutual properties. Things can be classified into classes or natural kinds. This section attempts to map these ontological concepts to UML constructs. By doing this we narrow the possible uses of UML constructs. This is reflected in the derived rules. We begin our discussion with the most basic concept, that of a thing.

4.1.1 Things

A primary concern of every modelling endeavour is the identification of the basic structure of the domain. In UML, the modeller needs to decide what to model as an object and what not to model as an object. Questions such as "is an employee an object?" or "are skills or transactions objects?" are of fundamental importance.

Because every BWW-thing is an entity in the world, we propose that a BWW-thing is equivalent to a UML-object. The inverse is not necessarily true: Not every UML-object is equivalent to a BWW-thing. Entities such as "location", "jobs", "orders" [1] are not things in the ontological sense but are still often modelled by UML-objects. If we want to assign the ontological semantics of a thing to a UML-object, we must follow rule 1:

Rule 1 *Only substantial entities in the world are modelled as objects.*

Fig. 4.1 is an example UML class diagram to show the implications of our rule. The model is taken from (Fowler and Kendall, 2000) and described as an anlysis level model, not a design model. It depicts a situation typically found in object-oriented models. Note that according to rule 1 the UML objects "Order" and "OrderLine" should *not* be modelled here as they have no substantial counterpart in the real-world. Hence, we must find an alternative description for entities such as "Job", "Order", etc.

4.1.2 Properties

Having identified what to model as an object, and what not to, we find that many of the items that we rejected as objects are properties of things. For while an employee should be modelled as an object, skills of employees should not, nor should addresses of employees as these are not substantial things. Instead, they are BWW-properties of things and we suggest they should be modelled as UML-attributes.

In ontology, every thing possesses properties, which may be either intrinsic, possessed by the thing alone, or mutual, joint properties of two or more different things. We propose that all properties find their equivalent

[1] We are not interested in the physical manifestations, such as an order form. These are merely descriptions of an order.

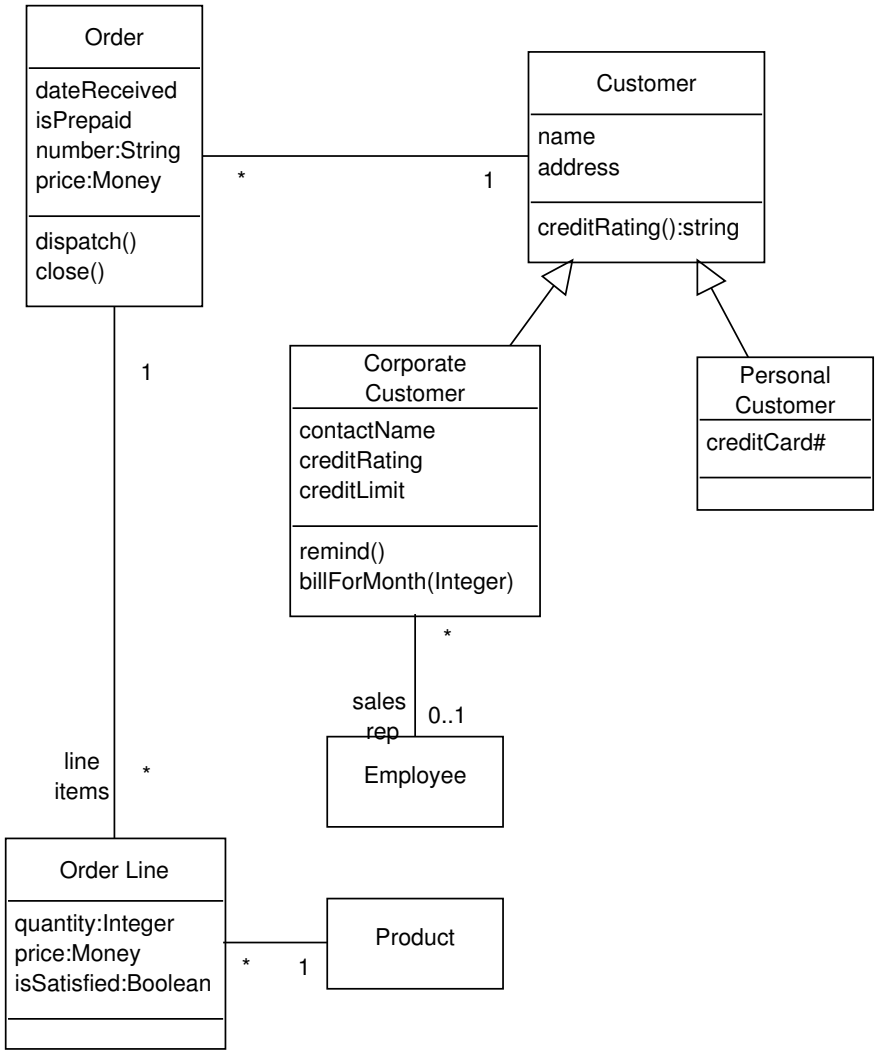


Figure 4.1: Example UML class diagram without ontological semantics (Fowler and Kendall, 2000)

in UML-attributes. The following paragraphs examine intrinsic and mutual properties.

Intrinsic Properties

We propose that BWW-properties of things are equivalent to UML-attributes of an object. Thus, when using UML for conceptual modelling, we suggest following rule 2:

Rule 2 *Ontological properties of things must be modeled as UML-attributes.*

One implication of this rule is a clear distinction between attributes and objects which reflects the clear distinction between BWW-properties and substantial things (see also Wand *et al.*, 1999). For example, while an address is a property of a person, it is not part of a person. The entity "Person" in the information system is a representation of a real person and as such can be compositionally associated in the IS with another representation, "Address". In the real world, a person is the physical entity but an address is a conceptual construct and they cannot be compositionally associated. Rules 1 and 2 together lead us to propose the following:

Corollary 1 *Attributes in a UML-description of the real world cannot refer to substantial entities.*

Consider for example a person that has a language skill. Because a skill is a conceptual entity, i.e. it is not substantial, it cannot be modeled as an object but must be an attribute of the person [2]. Ontologically, a skill is a property of a person.

The above rules relate model elements to elements of the real world and can therefore not be formally expressed in the meta-model or in OCL.

Mutual Properties

The results regarding the representation of properties as attributes are not restricted to intrinsic properties but should be applicable to all properties.

[2]Note that in IS design, a skill is often modelled as an object or entity, but this must be the second step in the IS analysis and design process, after the description of the real world.

Hence, mutual properties must also be represented by UML-attributes. This section will discuss mutual properties in detail.

The BWW-ontology distinguishes intrinsic properties from mutual properties. The former are possessed by a thing alone while the latter are properties of two or more things. Since all properties must be represented by attributes (rule 2), we propose that attributes of 'ordinary' UML-classes represent intrinsic properties while attributes of association classes represent mutual properties. Recall that mutual properties are properties that are shared between two things, e.g. the salary of an employee of a company or the tuition fee of a student of a university. These properties can only be attributed to two or more things jointly, not to a single thing.

A special case of an attribute in UML is an attribute of an association class. At this point we are not concerned with associations themselves and defer that topic for later discussion (Sec. 4.2.7). It suffices to introduce associations as some kind of 'connection' between classes, i.e. an association represents some semantic relationship between the objects of the associated classes.

We propose that association classes represent bundles of mutual properties, each association class attribute represents a mutual property. E.g. the property "Order volume" represents a mutual property between a supplier and a customer, the property "Salary" represents a mutual property between an employer and an employee. Note that we specifically do not make any interpretation of the association construct itself yet.

Rule 3 *Sets of mutual properties must be represented as attributes modelled with association classes.*

With this interpretation of a bundle of mutual properties, an association class *cannot* represent or describe substantial entities because in that case its attributes would be intrinsic ones of a substantial thing.

Corollary 2 *An association class cannot represent substantial entities or composites of substantial entities.*

In the following we show that this is not a limitation. We examine three different semantics of association classes as they are commonly found in object-oriented models. Based on the above ontological mapping we suggest rules for how these three cases must be treated to conform to our proposed ontological semantics. These rules follow from rule 3 and corollary 2.

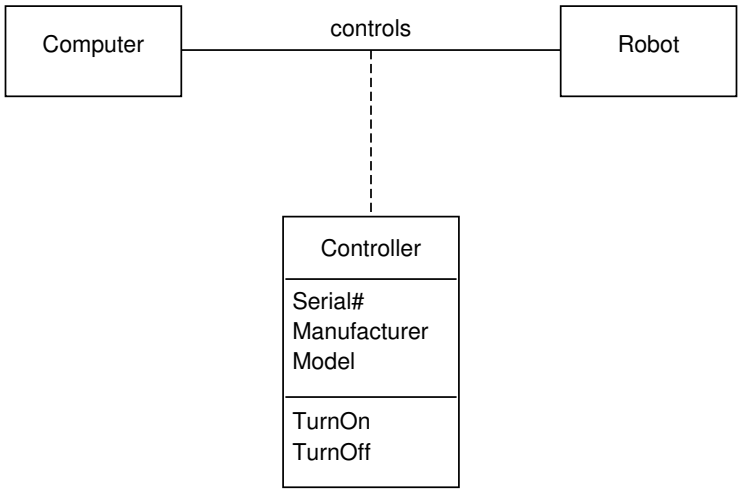


Figure 4.2: A substantial association class in UML

1. The association class is intended to represent a substantial entity.

   Figure 4.2 illustrates this case. A computer controls a robot via a controller device. This device has attributes and operations. In this case, the intended instances of the UML-association class are substantial entities and can thus be described as regular classes. The association class attributes are intrinsic properties of the controller device and should be modelled as such. This motivates the following rule:

   Corollary 3 *If an association class of an n-ary association is intended to represent substantial things, the association should instead be modelled as one with arity (n+1).*

   Fig. 4.3 shows this transformation using the above example. The controller association class is reinterpreted as a participant of the now ternary association. It has intrinsic attributes.

2. The intended interpretation of the association class can be identified as a composite or aggregate.

   Sometimes an association class represents emergent properties of a composite. When this is the case, the UML model should reflect this composition by showing an aggregation relationship. Figure 4.4 shows

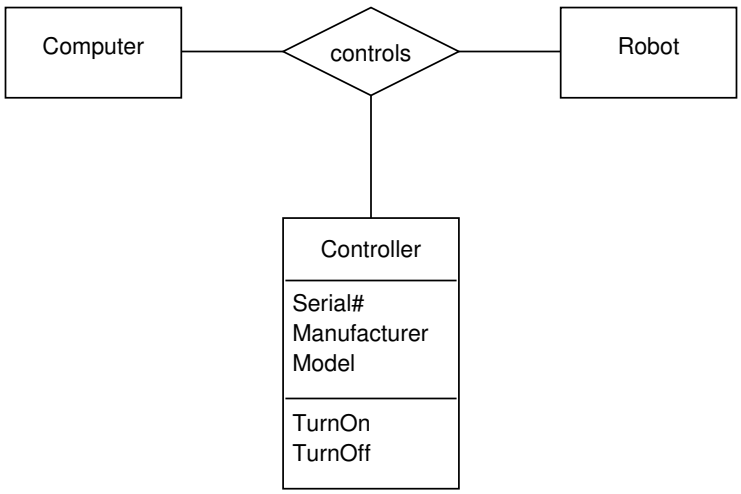


Figure 4.3: A reinterpreted substantial association class

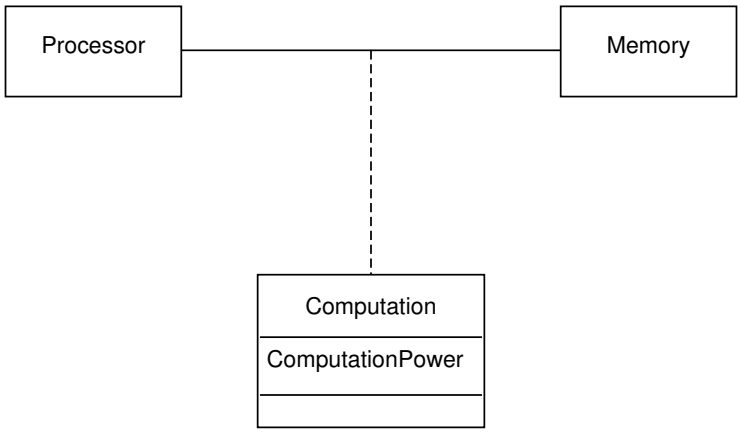


Figure 4.4: An association class represents a composite

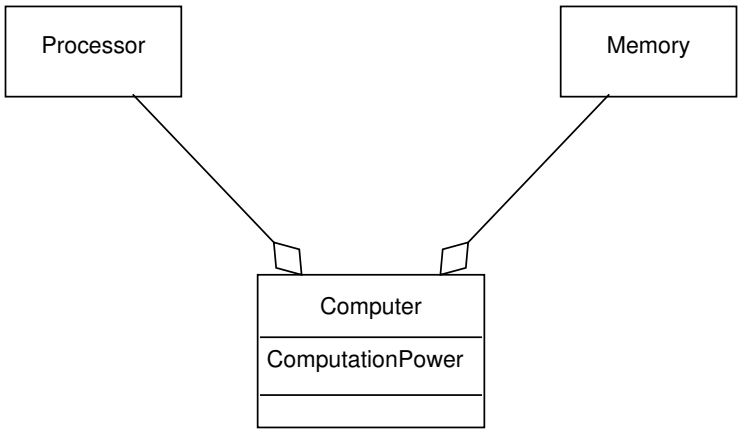


Figure 4.5: Reinterpreting an association class as a composite

an example. A processor and memory are associated to perform computations. This association has an attribute of computing power. This should instead be modelled properly as a composite with emergent properties, as depicted in Fig. 4.5.

Corollary 4 *An association class representing a composite must instead be modelled as a composite with attributes representing emergent intrinsic properties.*

3. The association class is intended to represent a set of mutual properties.

   Consider the situation depicted in Fig. 4.6. This third case reflects the ontological mapping that we have proposed above. A "Job" is not a substantial thing and hence cannot be modelled as a class. However, the attributes of the association class "Job" are mutual attributes of the company and the employee. They could be rewritten as a set of binary mutual properties:

   ```
   { JobTitle(Company, Worker)
     Salary(Company, Worker)
     StartDate(Company, Worker) }
   ```

   Additional laws are required to ensure that these properties occur together.

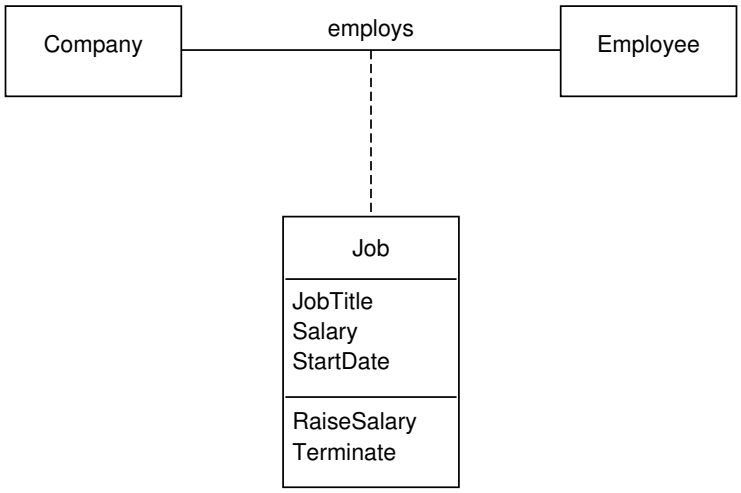


Figure 4.6: A conceptual association class in UML

Of the three possible intended meanings of an association class one already conforms to our ontological and the other two should not be modelled as association classes because their attributes represent either intrinsic properties or emergent intrinsic properties of composites.

Consequences and Characteristics of Association Classes Recall that in UML, an association class possesses all the characteristics of a class as well as that of an association. This means that in UML an association class can be generalized, participate in further associations, and define operations and methods. The next paragraphs will examine whether these characteristics are still attributable to UML association classes with our ontological interpretation.

Ontologically, all change is tied to things. There can be no change without a thing that changes. In light of the fact that association classes do *not* represent substantial things, we propose that an association class as a set of mutual properties cannot possess methods or operations nor can it be associated with a state machine specifying any behaviour.

Corollary 5 *An association class cannot possess methods or operations.*

We can express this in OCL as follows:

(4.1)

```
context AssociationClass inv:
self.feature
  ->select(f | f.oclIsTypeOf(BehaviouralFeature))
  ->size()=0
```

Corollary 6 *An association class cannot be associated with a state machine.*

(4.2)

```
context AssocationClass inv:
self.behaviour->Size() = 0
```

Moreover, since an association class attribute represents a mutual property, every association class must possess at least one attribute. While an empty set of mutual properties is still a set and thus technically satisfies corollary 3, it is ontologically meaningless. Hence we propose rule 7:

Corollary 7 *An association class must possess at least one attribute.*

(4.3)

```
context AssociationClass inv:
self.feature
   ->select(f : feature | f.oclIsKindOf(Attribute))
   ->size() > 0
```

Furthermore, as properties in ontology cannot themselves possess mutual properties with other properties or things, an association class in UML must not be associated with any other class.

Corollary 8 *An association class must not be associated with another class.*

(4.4)

```
context AssociationClass inv:
self.specifiedEnd->size() = 0 and
self.association->size() = 0
```

Since properties themselves cannot be generalized, an association class cannot be generalized or specialized.

Corollary 9 *An association class must not participate in generalization relationships.*

(4.5)

```
context AssociationClass inv:
self.child->size() = 0 and
self.parent->size() = 0
```

There seem to be numerous examples of property generalization: Having more than zero course credits generalizes the property of having three course credits, having ordered two items is generalized by the property of having ordered some number of items, having borrowed one book is generalized by the property of having borrowed books.

On first sight, the concept of precedence of properties appears to be a solution. It suggests that if the set of things possessing a property X is a subset of the things possessing property Y, then Y is said to precede X, formally: $Y \prec X$. However, this is not always a generalization. For example, the property of having items on order with a supplier is preceded by the property of having good credit with that supplier. However, having good credit is not a generalization of having items on order. Thus, property precedence is not equivalent to property generalization.

What about the above examples? (Bunge, 1977, pp. 68, 125) suggests that while attributes that represent properties can be dichotomized (e.g. borrowed one book, borrowed two books, borrowed three books), such dichotomization does not imply that the represented properties are dichotomous. Moreover, attributes that serve as state functions (see Sec. 5) should reflect properties in general such as 'number of books borrowed'. Hence,

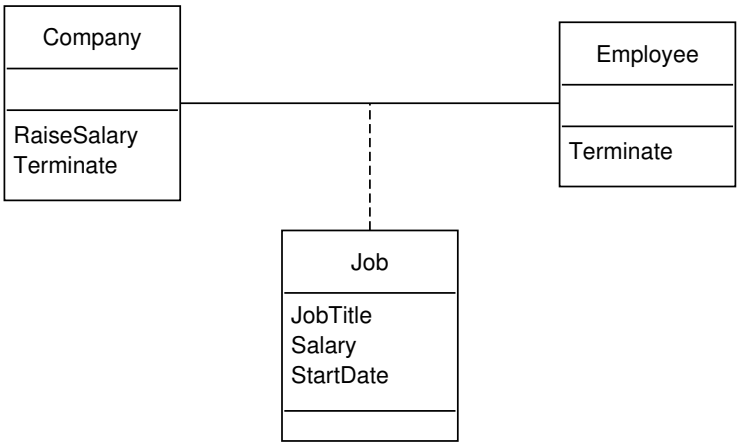


Figure 4.7: Association class and operations in UML

UML-attributes that represent properties should not be dichotomized unnecessarily or artificially.

What about properties such as 'can move' and 'can fly'? Clearly flying is a special way of moving. These attributes do not represent properties but potential change in things. Both, moving and flying are activities, not properties. Thus, they will need to be examined in our section on change (Sec. 5). Abilities should not be modelled as attributes.

In general, precedence of properties in the BWW-ontology is a law. We will later see that law statements are represented by UML constraints (see Sec. 6.1.1). Thus, we suggest to use UML-constraints to express general property precedence.

Recall that corollary 5 proscribes that association classes be assigned operations or methods. How then do mutual properties change?

Given that properties represented in association classes are mutual to the participants of the association, they may be changed when the things represented by the participant classes themselves undergo changes. Hence, we propose that methods and operations intended for association classes should be modelled with the participant classes. In the example of Fig. 4.6, the operation "RaiseSalary" should be modelled with the company and the operation "Terminate" may be modelled with either the company or the employee or both, depending on the real-world situation. Fig. 4.7 shows

this interpretation.

Rule 4 *If mutual properties can change quantitatively, methods and operations that change the values of attributes of the association class must be modelled for one or more of the classes participating in the association, objects of which can effect the change, not for the associations class.*

Mutual properties usually occur together. For example, some mutual properties arise out of interaction, e.g. after a customer and a car rental company interact, there exist a number of mutual properties such as 'DateOut', 'DateDue', 'BalanceDue', 'DepositPaid' etc. We suggest that sets of mutual properties that arise out of one interaction be modelled in a single association class. On the other hand, there exist situations in which unrelated sets of mutual properties exist. We suggest that each association class expresses related concurrent properties and that different association classes be used if sets of properties are not concurrent:

Rule 5 *An association class represents a set of mutual properties arising out of the same interaction.*

As a result, different association classes should be used for sets of mutual properties that arise out of different interaction. Consequently, two classes may be linked by more than one association. For example, the interaction of a customer ordering an item that gives rise to mutual properties such as 'Payment Balance', 'Expected Delivery Date', etc. between the customer and the supplier. Consider another interaction when the customer returns a defective product for warranty. Mutual properties such as 'Return Merchandise Number', 'Fault Description' etc. arise. These two sets of properties should in UML be modelled as two separate association classes between the company and the supplier. We suggest further that in cases where the interaction is modelled in other parts of the model (see Sec. 6) that the name of the association class refer to this interaction.

Mutual properties and scope of model It is often the case that mutual properties are properties of two or more things, some of which are beyond the scope of the model. In the simple case, a mutual property of two things, one of which is beyond the scope of the model, must then be modelled as an intrinsic property of the other thing. In doing so, the set of other things that the property is mutual with becomes part of the co-domain of the intrinsic property.

As an example, a student that has earned different degrees from different universities, e.g. a 'BSc' from UBC, a 'MSc' from UofA, a 'MBA' from UToronto and a 'PhD' from UVic. If the other schools form part of the model, an association class should be modelled with an attribute 'DegreeEarned'. However, if the other schools are beyond the scope of the model, but the degrees earned are not, we must instead model an intrinsic property by two attributes, 'DegreeEarned' and 'School' (or a single multi-valued attribute).

For the remainder of the discussion, the properties of any thing are represented by the union of attributes defined for a class and the attributes of association classes which that class participates in. Expression (B.4) defines this in OCL. As we discuss aggregation in subsection 4.1.3, expression (B.4) in Appendix B already includes as properties all properties of parts of aggregates.

4.1.3 Composition and Aggregation

Often, the modeller is confronted with the fact that different things are combined in some way to yield other things. However, not every combination of things is meaningful. This section will examine composition of things and develop guidelines to help ensure ontologically meaningful models.

The composition relationship in the BWW-ontology defines composite things as being made up of parts. In contrast to UML, it makes no assumptions as to any ownership or any restrictions on these part/whole relationships.

UML distinguishes between composition and aggregation. These differ along two semantic dimensions: In a composition relationship, the parts of the composite are existentially dependent on the composite and also cannot be part of any other composite. In an aggregation, the parts can exist independently of the aggregate and can also, at the same time, be parts of multiple aggregates. The aggregation semantics are equivalent to the ontological notion of composition, whereas the more restrictive UML-composition has no ontological counterpart.

Rule 6 *A composition relation must not be modelled.*

This can be expressed in OCL by proscribing the value "composite" as attribute of an AssociationEnd:

(4.6)

```
context AssociationEnd inv:
self.aggregation <> "composite"
```

In some cases it may the case that individuals change behaviour or lose or acquire properties if they become part of an aggregate. These changes may occur when they become part of the aggregate. Examples of this are the behaviour of team members as part of teams, employees as part of companies etc. Once could reasonably argue that team members 'disappear' once they leave the team. However, on closer examination, the team members do not disappear or get destroyed: Changes in properties or behaviour lead to sub-classification, a topic that will be examined closer in Sec. 4.2.3 in connection with multiplicities of associations.

Our ontology distinguishes between hereditary and emergent properties of composites. The former are properties of some part that the composite also exhibits, while the latter are properties of the composite that none of the parts exhibit. Emergent properties are defined over properties of parts and are thus a function of the properties of the parts. Take for example a composite of two parts, A and B, each with one property, e.g. u_1 and u_2. We can define the emergent property k as a function of the properties of the parts:

	$u_1 = 0$	$u_1 = 1$
$u_2 = 0$	$k = 0$	$k = 2$
$u_2 = 1$		$k = 1$

The property k is a function of the properties of the components A and B of the composite and can be reduced to these, but it is not exhibited by either component alone.

In the BWW-ontology, a composite must possess at least one emergent property, otherwise there exists not a composite but only a set of things (parts). In other words, a composite things must be more than simply the 'sum' of its parts. This leads us to propose rule 7 as a specific version of rule 8 below for composites:

Rule 7 *Every UML-aggregate must possess at least one attribute which is not an attribute of its parts or participate in an association with association class.*

Note that under attributes we understand attributes that represent both intrinsic and mutual properties, as defined in expression (B.4). Hence, we can formally express the above rule by the following OCL expressions, making use of expression (B.3) in Appendix B:

(4.7)

```
context Class inv:
self.association->exists(a | a.aggregation="aggregate")
 implies
self.allNonPartProperties->IsEmpty() = false
```

Consider again the example depicted in Fig. 4.4 of the processor and the memory. Only the fact that together the processor and memory give rise to a property makes the composition meaningful. Otherwise, the processor and the memory would simply be two things that happened to be close together. Our rules ensure that only meaningful compositions of things are modelled as an aggregate in UML.

4.2 Interpretation Mapping

The previous section examined the basic ontological concepts of the static structure of the real world: Things, properties and composition of things. This section will examine UML and object-oriented concepts and constructs. With the previously made representation mapping in mind we identify their ontological interpretation and explore the consequences of the representation mapping.

4.2.1 Class

In UML, a class is defined as a *description* of a set of objects which share the same attributes and operations. The role of classes in UML as primary construct is that of templates or 'object factories'. Objects depend on classes and cannot exist without them. Objects are created from class templates, i.e. an object as a class instance possesses all methods and attributes defined for its class. A UML-class is a description of a number of objects.

A BWW-natural kind is the set of things that adhere to the same set of

laws and in turn exhibit the same type of behaviour. A natural kind is a set of things, *not* a description or a template of or for a set of things. Sets differ semantically from classes in that they are defined over pre-existing things and do not exist without the things that comprise them.

Thus, UML-classes and BWW-natural kinds are fundamentally different, hence we refrain from mapping them onto each other. However, the BWW-ontology provides the concept of a functional schema. Such a schema is a description of the properties of a thing or a set of things with like properties. We propose that a UML-class is equivalent to a functional schema describing the entities that form the corresponding BWW-natural kind.

Because all things possess properties and adhere to some laws, the functional schema must reflect this and so we suggest the following rule:

Rule 8 *All UML-classes must possess at least one attribute or participate in an association.*

It is however possible that things possess attributes inherited from their parts. They are then defined as aggregate classes which participate in aggregations. In that case, rule 7 is applicable, which in turns yields rule 8.

The following OCL expression is an invariant on the Class construct which formalizes this rule:

(4.8)

```
context Class inv:
self.allProperties->IsEmpty() = false
```

Another important aspect of classes in object-oriented approaches is that of an 'object factory', i.e. classes serve to 'instantiate' or create objects according to the class template. Since in the BWW-ontology things cannot be created or destroyed, this aspect of a class has no ontological equivalent (but see Sec. 4.2.4).

4.2.2 Object Identity

Object identity, while not explicitly modelled in UML, is an important aspect in object-oriented techniques. In our ontology, things are identified

through their unique set of property values and there exists no special identification criterion or identifier. An Object ID as a special attribute has no ontological equivalent in the real world is thus excessive.

Rule 9 *Object ID's must not be modelled as attributes.*

Instead the modeller must determine a combination of attributes that uniquely identify an object[3]:

Rule 10 *The set of its attribute values must uniquely identify an object.*

The following OCL expression specifies this constraint:

(4.9)

```
context Class inv:
self.Instance->forAll(i1, i2 :  |
     i1.allProperties().slot.value =
     i2.allProperties().slot.value
implies
     i1 = i2)
```

In the context of information systems *design* it may not be desirable to include all identifying attributes. In this case they may be summarized or aggregated in the form of an artificial object identifier. This identifier thus represents all those properties of things that ensure its uniqueness. However, in the *conceptual* model, the identifying attributes should be modelled.

4.2.3 Multiplicities

UML allows the modeller to assign multiplicities to attributes and association classes. Because multiplicities express different semantics in each case, they are dealt with separately, beginning with multiplicities of attributes.

[3] In this respect relational data models, where the modeller defines a set of attributes whose values uniquely identify an entity, are closer to the BWW-ontology than object-oriented techniques which expressly rely on object identifiers.

Attributes

We begin with an important observation: Since attributes represent properties by being assigned some value at some time, these values reflect a property (Bunge, 1977, p. 119f). Hence, every attribute must have a value:

Rule 11 *Every attribute has a value.*

(4.10)

```
context Attribute inv:
self.AttributeLink->isEmpty() = false
```

Special values such as 'NULL', 'NIL' or 'void' are sometimes used to indicate the lack of a value. This contradicts rule 11. Moreover, these special values are not elements of the co-doamin of any real world property. Hence we proscribe their use as an attribute value in any model[4].

If there is no multiplicity assigned to an attribute it represents a simple property with implicit multiplicity of one, the domain of which is a set of values. In general, the co-domain of any property is a set. In the most general case, this set is an element of the powerset P of another set. Technically, the empty set ϵ is also a member of any powerset. We suggest that attributes with a declared multiplicity of k can take for a value all elements of P which consist of k element. It is of course meaningless to assign a multiplicity of zero to an attribute, as that attribute would only be able to take the empty set for a value. Not only is this ontologically meaningless, but it would also imply that the represented property is unable to change. This contradicts the BWW-ontology[5].

Attributes must correspond to substantial properties of a thing. Thus, while in theory attributes may be multi-valued or take as value the empty set, this may not correspond to the real-world situation. Furthermore, the modeller must take care to identify the meaning of the attributes in terms of the underlying properties. Consider the example depicted in Fig. 4.8. While the position in (A) is modelled with a multiplicity of two, this does not represent the true real-world situation. On closer inspection we find that the two values are not freely interchangeable, but that instead they

[4] See also (Wand *et al.*, 1999)

[5] Note also that the empty set ϵ is *not* equivalent to a 'NULL' value.

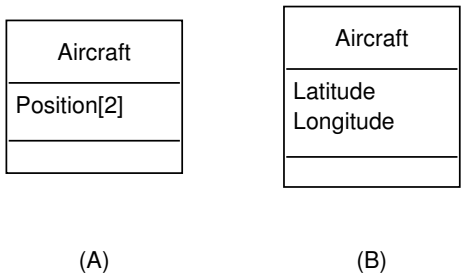


Figure 4.8: Multiplicity of Attributes

possess different meanings and should therefore be modelled as shown in (B). Thus, one criterion for when unary attributes are combinable into a multi-valued attribute is that the order or position is semantically irrelevant. This conforms to our set-oriented interpretation above, since a set too is by definition unordered.

Corollary 10 *Attribute multiplicities greater than one imply that the order of the different individual attribute value components is semantically irrelevant.*

Attributes may also be assigned multiplicities in the form of a range specifying lower and upper bound to express set-valued properties. The upper range can be unlimited, expressed by the '*' symbol. With our above interpretation and discussion, we suggest that such attributes can take values of a powerset that contain a variable number of values. Again, it must be noted that while the empty set ϵ is a member of any powerset it may not necessarily have an ontological interpretation in any given situation. Furthermore, corollary 10 applies to this case as well, suggesting the modeller must carefully analyze the represented properties.

For example, the declaration of a variable a with multiplicity three such as $a[3]$ while allowing 'NULL' values may seem to be equivalent to declaring $a[0..3]$ while *not* allowing 'NULL' values. In both cases, the attribute a can represent either zero, one, two or three values. However, we argue that the second representation better expresses the ontological real-world semantics, as it does not require use of special 'NULL' values. Still, it possibly violates rule 11 if it allows for value to be present and hence must not be modelled.

Note that the arguments in this section regarding multiplicities of attributes are valid for attributes of ordinary classes as well as for attributes of association classes, since both represent BWW-properties.

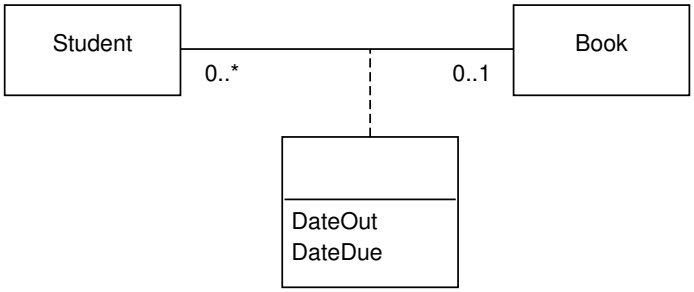
Association Classes

The semantics of multiplicities of associations, esp. association classes, are different than those of attributes. In contrast to attribute multiplicities, multiplicities assigned to association classes express the number of other things or objects a particular thing or object shares mutual properties with. Specifically, they do *not* express set valued attributes so that the arguments made above for UML-attributes are not applicable. Instead, multiplicities for association classes serve to support the acquisition of properties or loss of properties, as discussed in the following paragraphs.

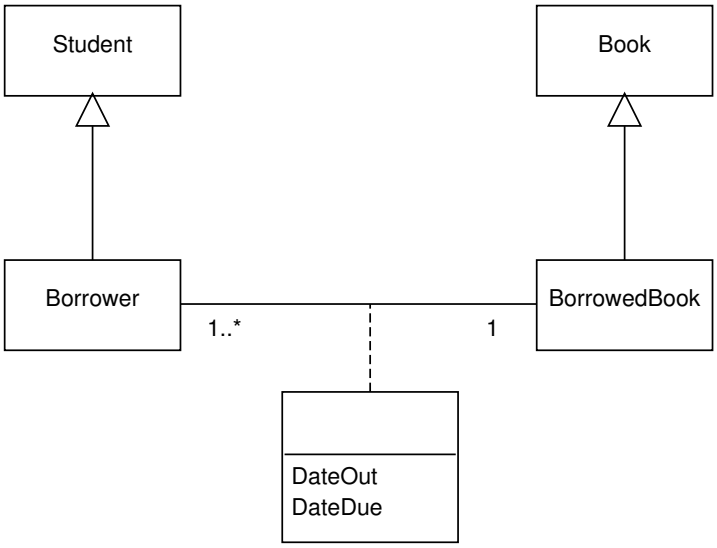
Wand *et al.* (1999) argue that a distinction should be made whether a thing possesses mutual properties or not. Using an example of a book and a student which may borrow it, they argue that instead of associating a student with zero or more books that she borrowed, the student should be specialized into a 'Borrower' which borrows one or more books while the more general 'Student' has not borrowed any books (Fig. 4.9). Their argument for this is that the model in Fig. 4.9(A) shows mutual properties (DateOut, DateDue) for the student, even if none are existent in the case where a student has not borrowed any books.

While this argument is sound for this example, it does not show the more general case. It could be argued for the above example that a similar distinction should be made between a borrower that has borrowed one book and a borrower that has borrowed two books, leading to yet another specialization relationship in the class diagram. This argument leads to an infinite number of classes to be modelled.

The fundamental reason why the distinction between possessing mutual properties with zero or one other things is generally considered more important than the distinction between possessing mutual properties with one or two things is that the acquisition of the first set of mutual properties (with the first other thing) also defines new behaviour for the class of things that acquire this first set of properties. Moreover, this additional behaviour is substantially identical for the subsequently acquired sets of mutual properties. In our example, the 'Borrower' can return a book, extend its loan period and exhibit other related behaviour. She can take the same actions



(A)



(B)

Figure 4.9: Optional properties and re-classification

for the second, third and any subsequent book that she borrows. The student who has not borrowed any books does not exhibit this behaviour. This distinction by behaviour is consistent with our interpretation of classes as functional schema of natural kinds, which are also characterized by the potential behaviour of their members (Sec. 2.3, also Sec. 4.2.1). This implies as a consequence that in certain domains it may be necessary to make further distinctions than the "zero" vs. "one or more" distinction usually made.

To summarize, a thing that either exhibits relevant additional mutual properties[6] or exhibits additional behaviour should be modelled as an object of a more specialized class. We therefore propose the following rules:

Rule 12 *Classes of objects that exhibit additional behaviour, additional attributes or additional association classes with respect to other objects of the same class, must be modelled as specialized sub-classes.*

Changes in natural kind of a thing correspond to changes in class membership of an object. UML does not provide a construct to express this. In this respect it is ontologically deficient. To complement rule 12, we suggest a mechanism for re-classification that employs the UML semantics of object creation and destruction[7]:

Corollary 11 *An object acquiring additional behaviour or properties must be destroyed as instance of the general class and created as instance of the specialized class that is modelled with the relevant operations or association classes.*

This rule leads the modeller to show the creation and destruction in sequence diagrams, which allow a time ordered view of object behaviour. More on object creation and destruction in Sec. 4.2.4.

Not every interaction leads to the acquisition of new behaviour. Hence, not every interaction is accompanied by changes in natural kind and thus not every interaction between two objects leads to specialization as described above.

Since all things possess at least one common property (Bunge, 1977, Post. 5.3), an important implication of the previous rule is that re-

[6] Relevance is subjective and depends on the purpose of the model.

[7] The fact that the object identifier change during destruction and subsequent creation is consistent with our ontology. If interpreted as an object's "name", the change agrees with Bunge's principle of nominal invariance (Sec. 4.2.4).

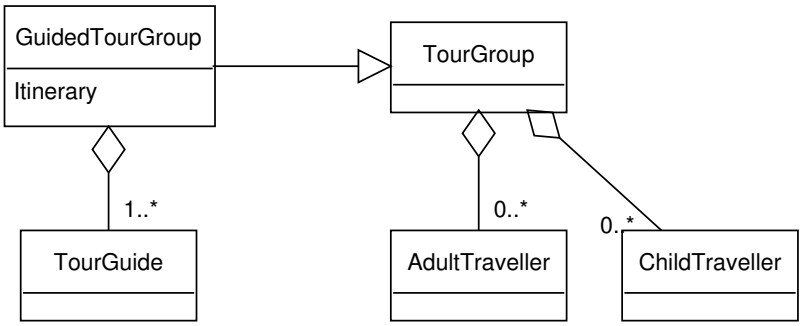


Figure 4.10: Example aggregation

classification, i.e. deep change, can only occur within a given generalization or specialization hierarchy. If the modeller identifies a need for qualitative change that does not fit a given hierarchy, she must search for and identify a common super-class to form such a hierarchy:

Corollary 12 *Re-classification occurs only within a generalization / specialization hierarchy.*

Aggregation Relationships

An even stronger argument concerning multiplicities and re-classification can be made for the case of UML-aggregation which models the ontological notion of composition. A thing cannot be a composite thing if it does not comprise at least two parts. Thus, we formulate the following rule:

Rule 13 *Every UML-aggregate object must consist of at least two parts.*

If the parts are homogeneous, this implies that the lower bound on the multiplicity must be two. If the parts are heterogeneous, no lower bound on each of the aggregation relationships can be inferred. Note in this context also rule 7 above, that requires aggregates to possess emergent properties.

As discussed above, the acquisition of properties can be through assembly, in which case the acquired properties may be emergent, i.e. an intrinsic property of the composite or aggregate. The example in Fig. 4.10 illustrates this. A tour group consists of at least two travellers. A guided tour group adds to this one or more tour guides. Note that in this case rule 13 does not

lead to a specific lower bound on any of the aggregation relationships, as the tour group and also the guided tour group are heterogeneous aggregates, consisting of adult travellers, child travellers and tour guides.

In analogy to re-classification in connection with acquisition of mutual properties, the acquisition of additional parts does not necessarily lead to re-classification of instances and specialization of classes. However, the acquisition of emergent properties must be modelled by specialization since property acquisition leads to different or additional possible behaviour of things (see above, also (Wand *et al.*, 1999)). Consider a tour group composed of five adults. Adding five children to the same tour group will not lead to a re-classification of the tour group. This is because no emergent attribute is acquired through the addition of the five children, whereas the addition of one or more tour guides leads to the acquisition of e.g. an 'itinerary' property and thus to a re-classification of the tour group as a guided tour group because a guided tour group can e.g. change its itinerary. We therefore propose the following rule, a special case of rule 12.

Rule 14 *An instance of a class that by virtue of additional aggregation relationships acquires emergent properties or emergent behaviour must be modelled as an instance of a specialized class which declares the corresponding attributes and operations.*

While the earlier rule 12 was derived based on acquisition of mutual properties and behaviour through interaction, rule 14 refers to acquisition of emergent intrinsic properties arising out of aggregation relationships. The mechanism suggested above to express re-classification using object destruction is also applicable in this case. More on object creation and destruction presently.

4.2.4 Object Creation & Destruction

Object creation and destruction have no direct equivalent in the BWW-ontology as things cannot be created or destroyed. Instead we relate these notions to Bunge's *principle of nominal invariance* (Bunge, 1977):

> *"A thing, if named, shall keep its name throughout its history as long as the latter does not include changes in natural kind - changes which call for changes of name."*(Bunge, 1977, Princ. 5.1)

Consistent with this principle, object creation and destruction have been related to changes in classification (see Sec. 4.2.3). This principle supports our interpretation of changes in classification through object destruction and creation. Changes in classification occur through acquisition or loss of a property. Two cases are common for property acquisition, corresponding to the two instances discussed for sub-classification above (rules 12 and 14).

1. The acquisition of the property is by composition or assembly and the acquired property is an emergent one.
2. The acquisition of the property is by interaction and the acquired property is a mutual one.

By analogy, object destruction occurs when an object loses a property which is necessary for membership in a certain class or kind. In Sec. 4.2.3 object creation and destruction were discussed with respect to changes in classification. A special case is the classification of an object as instance of a class when before it was an instance of a class that is not part of the model. In this case there is no object destruction being modelled that corresponds to the object construction. The following examples illustrate these ideas.

A set of bricks combined into a house 'creates' the new thing 'house' with the emergent property of 'NumberOfBedrooms'. This is object creation by composition. Altering the way the bricks are combined makes the house into an office building. It undergoes a qualitative change, losing the property 'NumberOfBedrooms' and acquiring the property 'NumberOfOffices'. A broken machine being fixed so that its parts start interacting again is an example of the second case. If we now assume that the house but not the bricks are part of our description of the world, then it would appear to us that a house had been created. Similarly, if we assume that 'house' but not 'office building' is part of our description, it would appear as if a 'house' had been destroyed. These examples motivate rule 15:

Rule 15 *Object creation occurs when an entity acquires a property so that it becomes a member of a different class.*

Corollary 13 *Object destruction occurs when an entity loses a property that is necessary for membership in a particular class.*

As discussed above, object creation and destruction usually occur together to suggest reclassification of an instance within the model, except when the 'source' class or the 'target' class is outside the scope of the model.

This interpretation of object creation and destruction is for purposes of conceptual modelling. In IS design, the interpretation and the rules for object creation and destruction may be different. For example, in the real world, a customer is not created, rather a person becomes a customer, once she acquires the property of having bought an item. In the IS we can however create a new customer record.

4.2.5 Class Attributes

UML allows the modeller to attach attributes to classes or, technically, declare the scope of an attribute to be the class. Thus, a class attribute is an attribute of the class of objects, not of an instance. The functional schema that we have mapped to UML-classes in Sec. 4.2.1 is not a substantial thing and hence cannot possess properties representable by attributes. Thus, we cannot ascribe attributes to it that would represent properties.

Instead, we propose to interpret a class attribute as representing a property of the composition of things that form the extension of the natural kind which is in turn described by the functional schema corresponding to the UML-class. Thus, the property is an emergent one and we suggest that the composition be modelled explicitly:

Rule 16 *Attributes with class scope should instead be modelled as attributes of an aggregate representing the objects of the class.*

No OCL expression can be given for this rule as it does not relate different meta-model elements. The rule instructs the modeller which of two modelling alternatives to use, both of which are consistent with the UML meta-model.

A typical example is the property "Number of Instances", that is often ascribed to a UML-class. This is a property of the collection or composite of all the things that are in the extension of the corresponding BWW-natural kind. A specific example is shown in Fig. 4.11. Whereas UML allows us to model a number of aircraft as in situation (A), ontologically we require that the class be made explicit by using composition as shown in situation (B).

Some object-oriented approaches argue that classes themselves are non-composite objects (see e.g. Opdahl and Henderson-Sellers, 2001). This allows the treatment of class attributes and class operations, such as the "new()" operation to create an instance, on the same footing as attributes

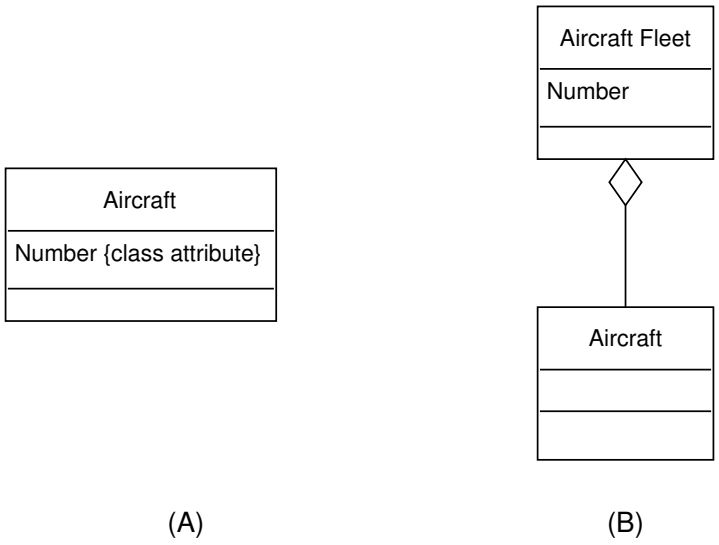


Figure 4.11: Class attributes

and operations of regular objects. However, such 'class objects' would not be creating things, instead they may assemble things or cause changes of classification through modification of things. This assembly interaction should be modelled independently and separately from the class in its role as description and templace of instances, e.g. by explicitly including a factory object or class in the model.

4.2.6 Abstract Classes & Generalization

In UML, generalization is realized by attribute and operation inheritance and method specialization. It describes a relationship between two UML-classes where a specific subclass has all the features of a more general superclass and may possess additional features. In the BWW-ontology the concept of generalization is very similar although generalization is not defined by inheritance but rather through the scope of properties or laws. Thus, a natural class defined using the same and additional laws as another natural kind can be considered as a specialized natural kind.

While more general classes in UML are often declared to be 'abstract', i.e. with no instances, the discussion of the BWW-ontology in section 2.2

showed that there are no empty natural kinds in ontology. There are no properties without things possessing them and there are no laws without things adhering to them.

Therefore, the declaration of an abstract class that is specialized implies that there exist no members of the more general class that are not members of one of the specialized classes. Hence, declaring a specialized class as abstract is equivalent to declaring the specialization as complete.

Rule 17 *If a class that is specialized is declared as abstract, the specialization must be declared to be 'complete'.*

The UML reference manual (OMG, 2001) also recognizes this connection between completeness of a specialization and abstract super-classes.

On the other hand, if a class is not specialized, there exist no sub-classes that its instances could be members of. Hence, all the instances are members of that class itself. Hence, it cannot be declared abstract:

Rule 18 *A class that is not specialized cannot be declared abstract.*

In OCL:

(4.11)

```
context Class inv:
self.isLeaf() implies not self.isAbstract
```

Acquisition of properties occurs when either emergent or mutual properties are gained. Hence, we require that specialized classes define more properties or more behaviour than the general ones:

Rule 19 *A specialized class must define more attributes, more operations or participate in more associations than the general class.*

(4.12)

```
context Class inv:
self.parent-exists()
implies
self.allPartProperties->IsEmpty() = false
or
self.feature
  ->select(f | f.oclIsTypeOf(BehaviouralFeature))
  ->size() > 0
```

This rule is related to rules 12 and 14 which state that if a class defines more operations or properties (represented as attributes or attributes of association classes), then it must be specialized. These three rules together now specify *necessary and sufficient* conditions for specialization.

With rule 19, we are also in a position to summarize specialization of classes. Any class that is a special class of a more general super-class, must declare any of the following:

- Non-inherited operations (representing intrinsic, emergent behaviour),
- Non-inherited attributes (representing intrinsic, emergent properties),
- Non-inherited participation in an association class (representing emergent mutual properties).
- Non-inherited participation in an aggregation relationship (which in turn requires non-inherited attributes, see rule 7).

4.2.7 Associations

There exist three kinds of associations in UML, of which only the 'ordinary association' is of interest here. Composition and aggregation associations are discussed in Sec. 4.1.3.

Section 4.1.2 provided an interpretation of association classes and their attributes while deferring treatment of associations. In UML an association class is a subtype or specialization of both a UML-association and a UML-class, i.e. an association class *is an* association. For the ontological

interpretation of UML-associations we need to briefly introduce the main results of the discussion on interaction, following below in Sec. 6.2.1.

Associations are employed in UML to enable message passing. Our discussion in Sec. 6.2.1 below will show that a stimulus, the instance of a message, travels along a link, the instance of an association. Hence, associations are necessary for interaction to occur *in the message-passing paradigm.* However, since message passing is a design related concept and there exists no equivalent in the BWW-ontology, we propose that associations are ontologically excessive (see Sec. 6.2.1). Hence, they should not be employed for conceptual modelling. Instead their use should be deferred until the IS design phase. We express this using the following rule:

Rule 20 *Every ordinary association must be an association class.*

(4.13)

```
context Association inv:
self.oclIsTypeOf(Association) and not
self->forall(ae : AssocationEnd | ae.aggregation
                                    = "aggregate")
implies
self.oclIsKindOf(AssociationClass)
```

One can argue that boolean mutual properties, ones that either exist or do not, should be modelled by associations. However, such a suggestion would lead to ambiguities since the same ontological concept, mutual properties, would be mapped to two different UML constructs, attributes and associations. In the interest of ontological clarity this should be avoided. Hence, even boolean mutual properties should be expressed using association class attributes.

This interpretation proscribes modelling any concept not related to mutual properties or interaction as an association. Specifically, mutual properties should not be modelled as associations but as attributes of association classes (see rule 3). As an example, the mutual property of 'being employed' should not be modelled as an association between an Person and a Company. Instead, an association class should be used with attributes such as 'Job Title', 'Salary', etc.

The following two examples, taken from (Wand *et al.*, 1999) demonstrate

our interpretation. Both of them are interpreted as associations by Wand *et al.* (1999).

A student attends a university

Ontologically, this is an example of interaction giving rise to mutual properties. Hence, it must be modelled as an association class that represents properties arising out of the interaction of enrollment. Examples of such properties are 'CreditsOnFile', 'Academic Standing' or 'Tuition Balance Owing'. It may be that the interaction is outside the scope of the modelled domain, i.e. in the past, so that the re-classification of e.g. a person as a student, is not modelled. If the interaction is part of the model, there must exist a class 'Person' of which 'Student' is a subclass. Only the students share mutual properties with the university and upon the enrollment interaction taking place, an instance of class 'Person' must be destroyed and re-created as instance of class 'Student' with the requisite mutual properties being modelled as attributes of an instance of the association class. Furthermore, the interaction itself must be modelled and is subject to the rules developed in Sec. 6.1.1.

A customer orders a product with product number XYZ

This is again an example of interaction giving rise to mutual properties between a customer and a supplier. It gives rise to mutual properties such as 'Quantity Ordered', 'Quantity Delivered' or 'Balance Owing'. Similar to the first example, the model may be different depending on whether the interaction itself is part of the modelled domain.

4.3 Summary

This section began by examining the central ontological concepts and mapped them to UML constructs. These include things, properties and composition. With this representation mappings in mind, we have interpreted object-oriented concepts such as classes, object identity, attribute and association multiplicities, object creation, class attributes and generalization / specialization. Table 4.1 shows a summary of the mappings.

The table shows that most of the basic ontological concepts have a UML equivalent and this mapping is a one-to-one relation or has been made so by

Ontological Concept	**UML Construct**	**Remarks**
Thing	Object	
Property	Attribute	
Intrinsic Property	Attribute of 'ordinary' class	
Mutual Property	Attribute of association class	
Emergent Property	Class attribute	
Functional Schema	Class	
Natural Kind	Aggregate object	Described by class
Composition	Aggregation	
	Composition	
Property inheritance	Attribute inheritance	
Re-classification		Explained by object creation and destruction
	Object Creation	Employed to express re-classification
	Object Destruction	dto.
	Object Identifier	Property values determine identity
	Association	

Table 4.1: Summary of Static Structure Interpretations

our proposed rules. On the other hand, there exist some UML constructs and concepts that have no direct equivalent in ontology. Some are examples of construct excess, such as object identifiers or composition, others can be explained in ontological terms without having a direct ontological counterpart, such as object creation and destruction. There also exist cases of construct deficits, such as the lack of a construct to describe changing class membership of objects. For these, we have found alternative expressions using existing UML elements, e.g. processes involving object creation and destruction.

Chapter 5

Change

The previous chapter examined the static structure of the world and laid the foundation for the present discussion of change. The BWW-ontology postulates that everything changes and that change is manifested in state transitions. As with the previous chapter on static structure, this chapter begins with the representation mapping followed by the interpretation mapping.

5.1 Representation Mapping

The central concept related to any change in the BWW-ontology is that of a state. Change is manifested through changes in the state of things. Hence, the representation mapping identifies the UML constructs corresponding to this ontological notion.

5.1.1 States and State Transitions

In the BWW-ontology, states are intricately tied to values of attributes. A state is the complete assignment of values to the state functions, i.e. a vector of the values of all attributes. There exist no states which are independent of attributes because properties express all the characteristics of a things. A state transition changes the state that a thing is in.

UML provides states and the semantics of state machines to specify discrete dynamic behavior. The UML manual (OMG, 2001) defines a state

loosely by some invariant condition, i.e. a condition that holds while the object is an that state [1]. State semantics are based on Harel state charts (Harel, 1988; Harel and Gery, 1996). However, states in UML are independent of attributes or properties and UML provides no mechanism with which to specify any such connection[2].

We propose that BWW-states are mapped to UML-states and consequently that BWW-state transitions are mapped to UML-state transitions. This mapping allows us to transfer the relationship between states and properties in the BWW ontology to UML by proposing the following rule:

Rule 21 *A UML-state represents a specific assignment of values to the attributes and attribute of association classes of the objects for which the state is defined.*

This rule may be used by the modeller to ensure inter-diagram consistency between state charts and class diagrams.

In the UML meta-model, an excerpt of which is presented in Fig. 5.1, there exists no explicit relationship between states and attributes or attribute links [3] [4]. Instead, objects in the current meta-model are instances that are modelled as being composed of such assignments. While this may be true in software design, for real-world considerations such a model confounds an object (instance), i.e. the thing existing in the world, with its state. We propose (Fig. 5.2) that this distinction be made and states be explicitly tied to attributes [5]. For this, we suggest that a class, a subtype of model element, has a number of state machines (at least one) specifying it's behaviour, depending on the model or functional schema used to describe

[1] While UML allows the association of a state machine and states with other classifiers such as use cases, it unclear in such cases what the states are states of.

[2] While previous authors have noted the connection between attributes and states (Booch, 1994; Coad and Yourdon, 1990; Jacobson, 1992; Rumbaugh *et al.*, 1991), none provide explicit rules or constructs for specifying this connection and UML does *not* support this connection.

[3] An attribute link is an instance of an attribute which has a value. A value in turn is an instance.

[4] Note that a state machine only contains a single state, the top state. Other states are contained recursively as sub-states. This is why the meta-model also allows states not to belong to any state machine; the association of states to state machines indicates only direct containment.

[5] Chapter 3 mentioned that Rosemann and Green (2000, 2002) developed a meta model that could serve to compare UML with the BWW-ontology. However, their model does not include states and state transitions.

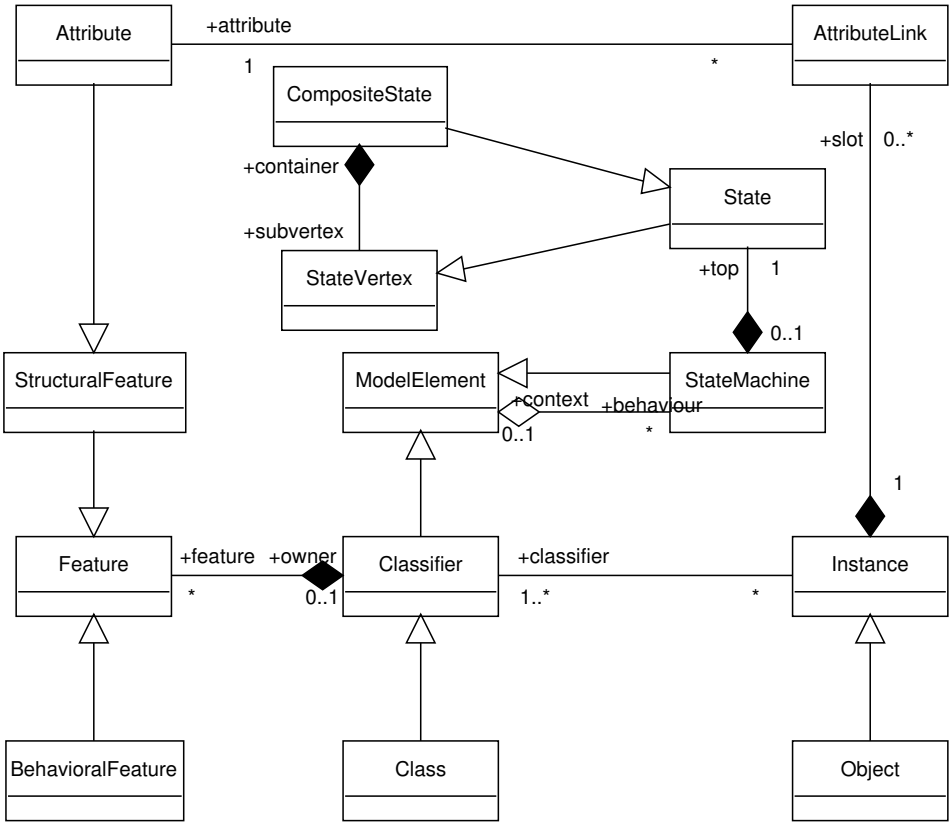


Figure 5.1: States and objects in the current meta-model

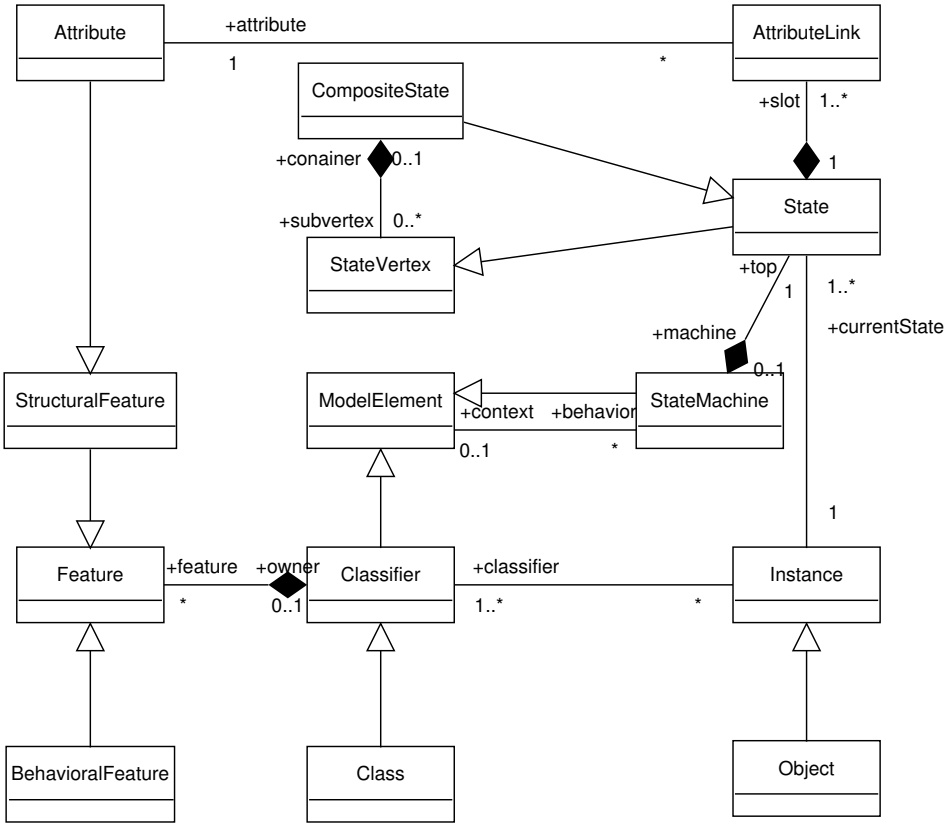


Figure 5.2: States and objects in the meta-model (proposed)

the behaviour:

(5.1)

```
context Class inv:
self.behavior->Size() > 0
```

OCL expression (5.1) augments (OMG, 2001, Sec. 2.12.3.5.1), which requires that a state machine be associated with either a classifier or a behavioural feature. We return to behavioural features in Sec. 5.2.5 below.

In our proposed meta-model, states are defined by a set of AttributeLinks. An AttributeLink is an assignment of a value to an attribute. The discussion in Chap. 4 showed that object attributes corresponding to properties of a thing are those that are defined for the object itself, inherited from super-classes as well as those defined for its parts (Sec. 4.1.2 and expression (B.4) in Appendix B). These properties therefore span the state space of a thing. Consequently, the corresponding attributes must be used to define the state of an object in UML:

(5.2)

```
context State inv:
self.Machine().context.allProperties()
   ->inludesAll(self.slot.attribute)
```

UML offers the construct of sub-states. These are states that are embedded in composite states in a single state machine. Hence, sub-states, when mapped back to ontology, must be states of the same state space. Hence, sub-states must be defined by the same set of attributes as their super-state.

(5.3)

```
context State inv:
self.oclIsTypeOf(CompositeState)
implies
self.subvertex
    ->select(sv | sv.oclIsTypeOf(State))
    ->forall(sv1, sv2 |
        sv1.slot.attribute = sv2.slot.attribute)
```

In the BWW-ontology, things can be in more than one state, depending on the model that is used to describe them. A thing is in one particular state at a point in time in each model. This ontological relationship must be mapped to UML. The proposed meta-model suggests that there exists more than one current state for an object (Fig. 5.2). In order to make sure that these correspond to different models in the ontology, these states must be defined by a different set of attributes, hence be part of a different state space:

(5.4)

```
context Instance inv:
self.currentstate->forall(cs1, cs2 : currentstate |
    cs1->attributeLink.attribute <>
    cs2->attributeLink.attribute)
```

An important ontological assumption is that state transitions are defined only within a single thing. This implies for UML that, because of the mappings made, state transitions can only occur between states of the same object:

(5.5)

```
context Transition inv:
self.source.Machine() = self.target.Machine()
```

In ontology, state transitions express changes in properties. Consequently, changes in properties may result in the change of a state, but all

changes of state are caused by changes to properties. Based on the mapping of states and properties, this ontological assumption can be transferred to UML and motivates the following rule:

Corollary 14 *A UML-transition must change the value of at least one attribute used to define the state space.*

Using the proposed meta-model (Fig. 5.2) this is specified in OCL expression (5.6):

(5.6)

```
context Transition  inv:
self.source.slot.value <>
  self.target.slot.value
```

To summarize, this section examined the ontological concepts of states and state transitions and mapped them to UML-states and UML-state transitions. In order for this mapping to hold, UML-states and UML-state transitions must obey the same principles as BWW-states and BWW-state transitions. The proposed mappings were used to transfer these ontological principles to UML and resulted in a number of modelling rules, which are formally represented by a change to the UML meta-model and a set of constraints expressed in OCL.

5.2 Interpretation Mapping

With the UML representation of ontological states and state transitions established in the previous section, this section proceeds to examine UML constructs that are closely related to state charts. The discussion begins by interpreting sub-states.

5.2.1 Substates

It is possible in UML to define a hierarchy of states. A composite state or submachine state may be refined as a state machine comprising substates and transitions among sub-states. A state may be refined into one or

more concurrently operating state machines, specified as orthogonal regions. Each orthogonal region in turn is a sub-state. The notion of orthogonality and concurrency is independent of that of composition or aggregation: "Orthogonality in statecharts is not intended for specifying components that correspond to different sub-objects." (Harel and Gery, 1996, p. 252).

The BWW-ontology does not contain the notion of sub-states. Instead, it requires the *complete* assignment of values to state functions for specification of a state. Hence, state functions and functional schema must be used to interpret sub-states. We will show by example that we can provide the same level of detail ontologically as found in UML state charts. From our discussion of the BWW-ontology in section 2.3 we know that a thing can be described by multiple different models or functional schemata, depending on which properties one is interested in. Consider the situation shown in the UML-diagram of Fig. 5.3. It depicts a state C refined using an initial pseudostate[6] and four sub-states. The system under consideration can be in state C and at the same time in different sub-states that leave the super-state C invariant. We can describe the same situation ontologically using a combination of functional schemata.

Assume that the composite state C is described by assignment of values to a state function S_C, e.g. $S_C = A$. A is an element of the co-domain of the state function S_C, e.g. some numeric value. The single state function S_C defines model M_1. The sub-states 0 and 3 and the unnamed sub-state composed of two orthogonal regions can be described by complete assignment of values to the two state functions S_C and S_{03}, e.g. $< S_C = A, S_{03} = X >$ for sub-state 0, $< S_C = A, S_{03} = Y >$ for sub-state 3 and $< S_C = A, S_{03} = Z >$ for the state composed of orthogonal regions. The two state functions S_C and S_{03} define model M_2. Since sub-states 1 and 2 are at the same level of abstraction and are concurrent we describe them by two different sets of state functions S_C, S_{03} and S_1 (model M_3) for sub-state 1 and S_C, S_{03}, and S_2 (model M_4) for sub-state 2, e.g. $< S_C = A, S_{03} = Z, S_1 = H >$ for sub-state 1 and $< S_C = A, S_{03} = Z, S_2 = I >$ for sub-state 2. In total we have described this situation ontologically using four different models, M_1 through M_4.

In general, states can be described by sets of state functions and their assignment. Thus S_C, S_{03}, S_1 and S_2 may be thought of as sets of state

[6] An initial pseudostate is the sub-state the system is in, when the super-state of that sub-state is entered. Pseudostates are constructs with a purely syntactic purpose and without ontological meaning.

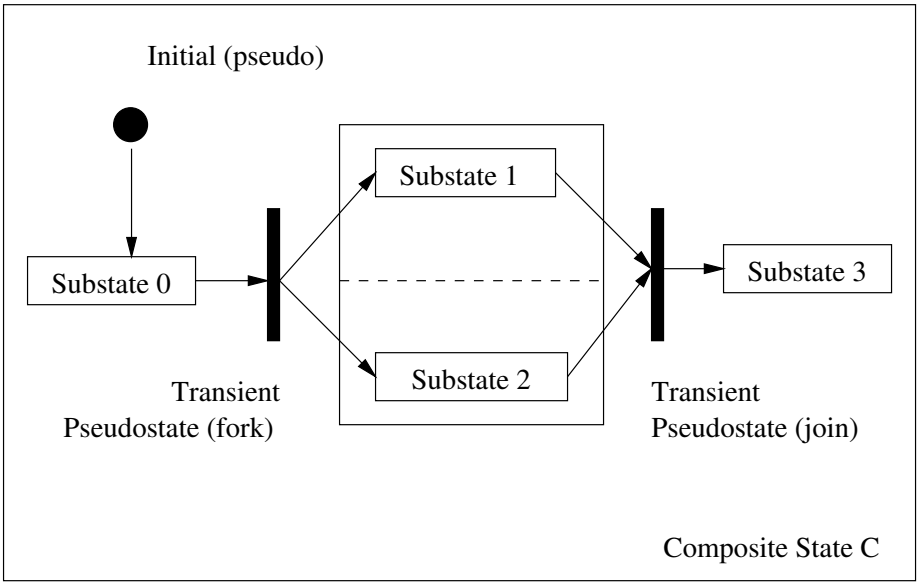


Figure 5.3: Composite states and sub-states in UML

functions, with the assigned values being vectors.

Our suggested addition to the UML meta-model (Fig. 5.2) incorporates these semantics by allowing that each model element such as a class has more than one state machine associated with it and that these include different states spanned by different sets of attributes.

The above example shows that whenever we encounter sub-states, the set of attributes used to describe them must be extended from the set used to describe the super-state. Whenever we move among sub-states on the same level, a set of attributes changes their values. From this discussion we can formulate the following rules:

Rule 22 *For every level of refinement of a state C, there must be an additional set of attributes in the class description or in participating association classes that change as the object transitions among the sub-states.*

This applies only if there are more than two sub-states of any such state. This is expressed formally in the following OCL expression which requires of any state that is a composite state that its definition include and extend the set of attributes that define the sub-states.

(5.7)

```
context State inv:
self.oclIsTypeOf(CompositeState) and
self.subvertex
  ->select(sv | sv.oclIsType(State))->size() > 1
implies
self.subvertex->select(sv | sv.oclIstypeOf(State))
  .slot.attribute
  ->includesAll(self.slot.attribute)
and
self.subvertex->select(sv | sv.oclIsTypeOf(State))
  .slot.attribute
  ->exists(a | self.slot.attribute->excludes(a))
```

Corollary 15 *For all immediate substates of a super-state, the values assigned to attributes describing the super-state are invariant and are equal to those defining the super-state.*

To express this in OCL we require that for all sub-states the attribute link values of those attribute links referring to attributes also found in the sub-state must be the same as the attribute link values of the latter attributes.

(5.8)

```
context State inv:
self->oclIsTypeOf(CompositeState)
implies
self
  .subvertex
  ->select(sv | sv.oclIsTypeOf(State))
  .attributeLink
  ->select(al | self.attributeLink.attribute
                ->includes(al.attribute))
  ->forall(al | al.value = self.attributeLink.value and
            al.attribute = self.attributeLink.attribute)
```

Hence, in the above example, the values assigned to $< S_C >$ when in sub-states 0 or 3 must be the same. Furthermore, by corollaries 22 and 15

concurrent sub-states require additional attributes which can concurrently take on different values, hence must be mutually disjunct:

Corollary 16 *Concurrent sub-states require mutually disjunct sets of additional attributes in the class description or in participating association classes.*

In OCL we ensure that the intersection of the attribute sets spanning the concurrent regions, excluding those defined for the containing state, is empty:

(5.9)

```
context CompositeState inv:
self.isConcurrent = true
implies
self.subvertex->select(s | s.isRegion=true)
  ->forall(s1, s2 |
     s1.slot.Attribute
     ->reject(a | self.slot.attribute->includes(a))
     ->intersect(
        s2.slot.Attribute
        ->reject(a | self.slot.attribute->includes(a)))
     ->isEmpty()
```

In conclusion, we have shown that we can benefit from linking the static structure description of a system to the dynamic concept of states without compromising the level of detail description that UML state charts provide. The proposed rules can help the modeller to ensure that class diagrams which define the attributes of objects are sufficiently detailed to support the desired behavioural characteristics in state charts.

5.2.2 Guard conditions

A common element of UML state charts are guard conditions. These are conditions on a state transition which have to be fulfilled in order for the object to transition between two states. The BWW-ontology does not contain the notion of guard conditions but allows the description of a state space and the possible events (transitions) in that state space. Hence, these ontological concepts must be used to interpret guard conditions.

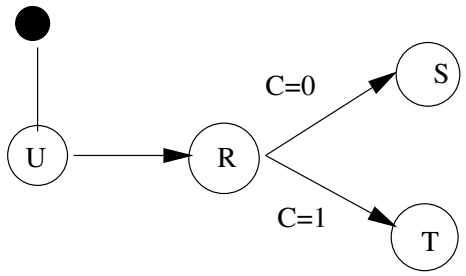


Figure 5.4: State chart with guard conditions

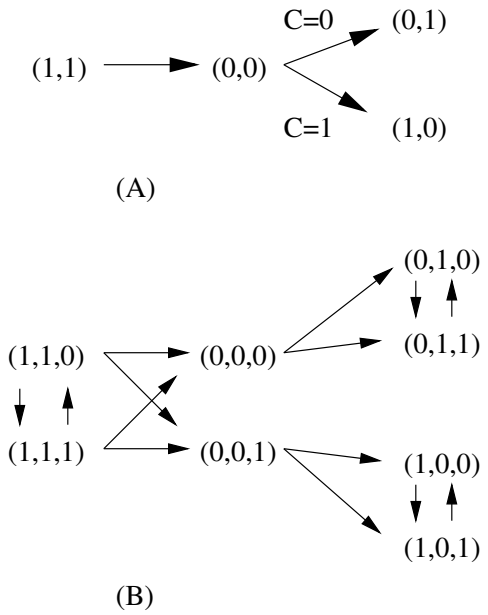


Figure 5.5: Reinterpreted State chart

Without loss of generality, assume an object with two binary attributes A, B. These span a state space of four states, namely $R = (0,0), S = (0,1), T = (1,0), U = (1,1)$ (Fig. 5.4). Note that the guard conditions on the state transitions (R,S) and (R,T) introduce an attribute C. Since the state space is spanned by all attributes of a particular model, this expands the conceivable state space to eight states: $R_0, R_1, S_0, S_1, T_0, T_1, U_0, U_1$ where the subscript denotes the values of attribute C. We can then transform Fig. 5.4 to explicitly include the expanded state space as depicted in Fig. 5.5. Fig. 5.5 (A) shows the initial state chart, equivalent to Fig. 5.4, whereas Fig. 5.5 (B) shows the transformed state chart. For demonstration purposes, we have written out the attribute assignments in this figure.

The resultant state chart does not contain an explicit guard condition. Instead the semantics of the guard condition are now modelled by the allowed state transitions.

Ontologically, a thing can be in only one current state in any given model. Guard conditions are employed to specify the conditional transition of an object from one state to exactly one of a number of others. Hence, we require that the guard conditions for all state transitions beginning with the same state and ending in non-concurrent region states be mutually exclusive, i.e. under no circumstances can two or more of them evaluate to true. Only then can it be guaranteed that the object representing the thing is not in two states at the same time, without these explicitly being concurrent sub-states. We propose the following rule to express this. Since the elements of guard conditions are not described in the UML meta-model, no OCL description can be given for this rule.

Rule 23 *Guard conditions on transitions from the same state to non-concurrent sub-states must be mutually disjunct.*

If this is not the case, the analyst must identify whether the two or more target states are actually concurrent regions of a single super-state. If that is the case, they should be modelled accordingly and rule 22 and corollaries 15 and 16 become applicable.

5.2.3 Action States

Action states in UML are states of an object during which the object undergoes a change. This is incompatible with the ontological notion that a

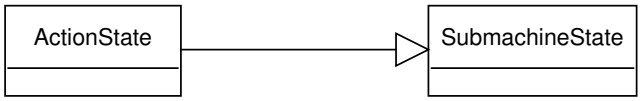


Figure 5.6: Action states as submachine states

state is an assignment of values to attributes at a specific instance in time. However, for an ontological interpretation of action states we can build on the interpretation for sub-states (Sec. 5.2.1). If there is change occurring while a thing is in a certain state, there must exist a state variable not employed for the state definition and that state variable may change. Hence, this can be represented in UML in the form of sub-states. We suggest to follow rule 24 when modelling action states.

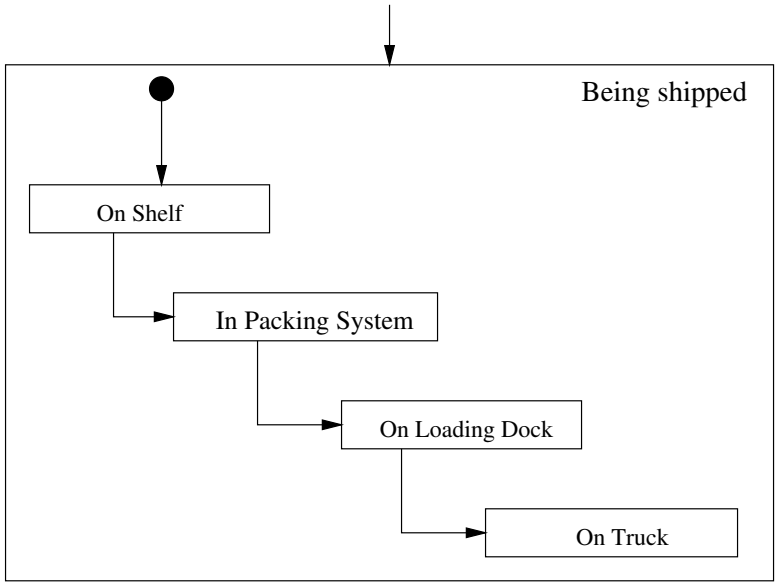
Rule 24 *Action states are super-states of a set of sub-states. The object transitions among these while in the action state. State charts must reflect this fact.*

Fig. 5.6 proposes a change to the UML meta-model which reflects this interpretation. In the current meta-model, action states are a specialization of simple states. We propose that they instead be interpreted as sub-types of submachine states and thus composite states.

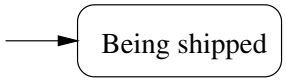
UML further allows an action construct to be associated with states. These actions are intended to represent the activity that is ongoing while an object is in that state. Based on the above discussion, there appears no need to assign such an action. Any activity can be described by the set of sub-states and the transitions among them.

Corollary 17 *States must not be associated with any actions. Sub-states corresponding to different models should be used instead.*

This translates to OCL as follows:



(A)



(B)

Figure 5.7: Action states as super-states

(5.10)

```
context State inv:
self.entry->Size()=0 and
self.exit->Size()=0 and
self.doActivity->Size()=0
```

An example of this interpretation is shown in Fig. 5.7. The situation in (A) is equivalent to that in (B). An item that is in the action state of "being shipped" undergoes state transitions while being in that state, e.g. from state "on shelf" to a new state "in packing system" etc. Since action

states often appear in activity diagrams, this rule ensures inter-diagram consistency between these and state charts. Because of the sub-classing of action states as submachine states, corollaries 15 and 16 are also applicable to action states.

5.2.4 Partitions

Partitions ("swimlanes") are employed in UML activity diagrams to group states or action states together. These partitions have no further semantics in UML, but are often featured prominently in activity diagrams. It is therefore important to examine them ontologically.

In UML, an activity diagram is a state machine and state machines describe the behaviour of the objects of a single class or the implementation of an operation (see Sec. 5.2.5 below). The elements of a partition are therefore interpreted ontologically as states and state transitions between states. Ontologically, states are defined for a given thing and state transitions are defined only between states of the same thing (see also Sec. 5.1.1). Given this ontological constraint, partitions *cannot* represent objects or classes of objects. Otherwise, state transitions that cross partitions would represent state transitions between different objects. This leads us to propose the following rule:

Corollary 18 *All states in an activity diagram must be states of the same object.*

No OCL expression needs to be given, as the meta-model ensures this rule through the specialization of StateMachine to ActivityGraph. We state this rule for the benefit of visually modelling activity graphs.

Recall that the attributes of an object include those of its parts. If states of what may appear to be different objects are included, the modeller or analyst must examine whether these different objects are not parts of a composite. For example, activity graphs are often used for business process representations Dussart *et al.* (2002). State transitions between two partitions, e.g. "Marketing Department" and "Production Scheduling Department" can be interpreted as state transitions within the single composite object "Company" or "Plant". In that case, this composite object must be explicitly modelling in a class diagram.

Corollary 19 *If the partitions of an activity diagram represent different objects, they must be part of a composite which is shown in the class diagram.*

5.2.5 Operations

The BWW-ontology provides no construct that is equivalent to methods or operations and it has been suggested that these are an implementation related construct, i.e. without relevance for real-world models (Parsons and Wand, 1991, 1997) and thus need not be interpreted ontologically. However, as they feature prominently in class diagrams, it is important that they be given some ontological explanation.

Whereas operations specify behaviour only in abstract terms without suggesting an implementation, a method provides just this implementation of an operation. As dynamics are expressed ontologically through states and state transitions, these concepts must be used to interpret methods and operations ontologically. We begin by examining the operation construct which is followed by a discussion of methods in Sec. 5.2.6.

Both state semantics and operations are related to the same ontological notion of change and so we require agreement between the two types of description. We begin the mapping with the observation that in the object approach, the *entire* range of behaviour is determined by the set of operations. Ontologically, all (quantitative) changes of a thing are describable by a series of state transitions among stable states. We therefore propose the following rules:

Rule 25 *The quantitative object behaviour (for each model) is* entirely *describable by top-level state chart* (SC_0)

We will see below, that this top-level state chart specifies the object behaviour *with respect to external events* (see corollary 22). In terms of the UML meta-model, this rule implies that for every class there must exist at least one top-level state chart that expresses the possible behaviour of the objects of that class. OCL expression (5.1) already requires the state chart. We must now ensure that this is a top-level state machine, i.e. not contained in others as a sub-state.

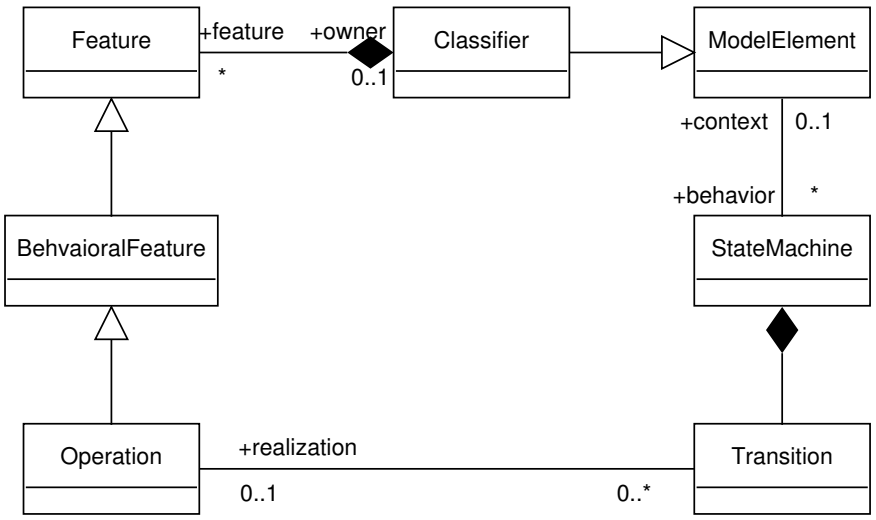


Figure 5.8: Operations and state transitions in the meta-model

(5.11)

```
context StateMachine inv:
self.context.oclIsTypeOf(Class)
implies
self.top.container->size()=0
```

Since the behaviour of objects is limited to the operations defined on the object, and the top-level state charts of objects describe this behaviour, we propose rule 26:

Rule 26 *All UML-transitions in SC_0 must correspond to an operation of the object which SC_0 is associated with.*

We propose that this rule be formalized as an addition to the UML meta-model (Fig. 5.8). An operation is the realization of one or more state transition. This allows a single operation to realize state transitions in two different models of the object. Conversely, a state transition may be realized by an operation of the object. The meta-model changes allows state transitions not to be realized by operations. This is necessary for example

when the state transition is not a state transition of a top-level state chart. The discussion below will show that methods themselves can be specified using state transitions and these method-internal state transitions need not themselves be realized by operations. Otherwise, this would lead to an infinite regress. We therefore express rule 26 in OCL as follows, making use of expression (B.11) in Appendix B:

(5.12)

```
context Transition inv:
if
   self.isTopLevel()
then
   self.realization->size()=1 and
   self.realization.owner=self.Machine().context
else
   self.realization->size()=0
endif
```

This specifies that if a transition is a transition of a top-level state chart specifying class behaviour, then there exists an operation that realizes it and this operation must be an operation of the same class that the state machine is specifying the behaviour of. Note that the OCL expression (B.11) in Appendix B defines also those transitions as top-level transitions that are contained in sub-states of a state machine specifying class behaviour.

We have suggested that the operations defined for an object express the entire range of possible behaviour. Since in our ontology everything must be able to change (either quantitatively or qualitatively), we propose rule 20:

Corollary 20 *Every object must have at least one operation.*

(5.13)

```
context Object inv:
self.classifier.feature
  ->exists(f : feature | f.oclIsTypeOf(Operation))
```

The BWW-ontology distinguishes among stable and unstable states.

Stable states are those states that are not left without some external interaction while unstable states are intermediate states that an object can transition out of by itself. In UML, an object remains in a particular state until an operation is invoked by some other object. As operations are realizations of top-level state transitions, this implies that top-level state transitions originate with stable states. Since all top-level state transitions originate from top-level states, we propose the following rule:

Corollary 21 *States in SC_0 are stable.*

The question then becomes how to represent the notion of stability in UML. UML contains no reference to stability or instability of states. Adding sub-classes to states in the meta-model is unsatisfactory, as this cannot express the semantics of external interaction through which stability is defined. However, it is possible to make use of the UML event construct.

In UML, external interaction is indicated through the event construct (see Sec. 6.2.2 below). An event may be associated with a state transition as the trigger of that transition[7]. This corresponds well with the ontological idea that some changes, i.e. state transitions, are externally induced and others are spontaneous. In the BWW-ontology, this is used to mark the difference between stable and unstable states. We can transfer the notion of stability to UML by noting that with our interpretation mapping UML-events serve as indicators of external action. It is therefore possible to identify those state transitions which are externally induced. An object is in a stable state, if all possible transitions out of that state have an associated UML event. If there exists at least one transition out of that state that does not have an associated event, the state is unstable as the object can undergo spontaneous, i.e. not externally induced, changes. OCL expression (5.14) defines a function that indicates whether a state is stable.

[7] UML models this as an aggregation. Ontologically, a transition does *not consist* of triggering events.

(5.14)

```
context State::IsStable() : Boolean
result =
if self.outgoing->forAll(t : Transition |
     t.trigger->Size()=1)
then
   true
else
   false
endif
```

We can now express corollary 21 in OCL as follows, making use of expression (B.12) in Appendix B:

(5.15)

```
context State inv:
self.isTopLevel()
implies
self.IsStable()
```

Because all the states in the top-level state chart are stable, the transitions must be associated with events:

Corollary 22 *All UML-transitions in SC_0 must be associated with a UML-event.*

Using expressions (5.14) and (B.11) in Appendix B we can express corollary 22 in expression (5.16):

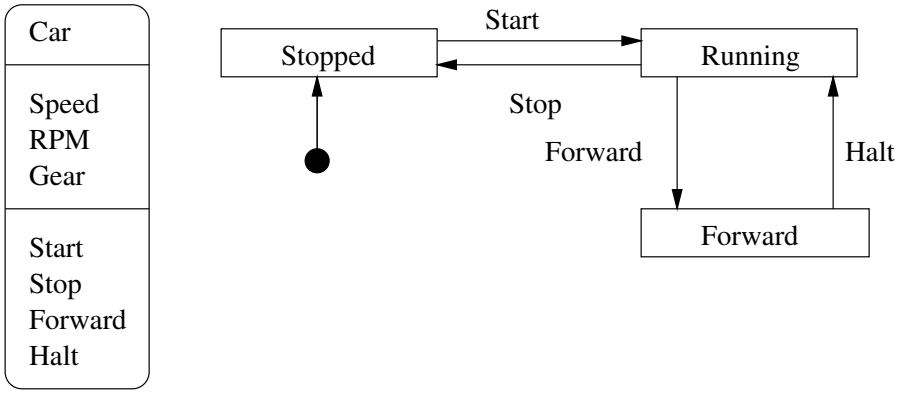


Figure 5.9: Class definition and state chart

(5.16)

```
context Transition inv:
if
   self.isTopLevel()
then
   self.trigger->size()=1
endif
```

The above rules and corollaries allow the modeller to identify operations that may have been missed in constructing the class diagram or identify operations for which there are no corresponding state transition. Such operations are redundant. Fig. 5.9 shows an example. Consider a car that can be in three states, stopped, (engine) running and going forward. Assume that the states of a car are defined as follows:

Stopped	$(Speed = 0, RPM = 0)$
Running	$(Speed = 0, RPM = 1000, Gear = 0)$
Forward	$(Speed = 50, RPM = 3000, Gear = 1)$

The class definition (A) is consistent with the top-level state chart SC_0 depicted in (B).

In addition to these rules, we remind the reader of rules 4 and 5 in the context of association classes that are applicable to operations as well.

The benefit of explicitly modelling the link between operations and state transition is to give the method designer some way of ensuring that the full state description of a system is realized. It also helps ensure that the methods and operations do not implement any change beyond that contained in the state description. Such additional behaviour could cause a faulty information system.

To summarize, both operations and state charts are linked to the same ontological concept of change and can be linked through state semantics. We conclude this section with an examination of the implications of qualitative change.

Operations and Qualitative Change

The *complete* behaviour of an object is defined by its operations and this behaviour must express the *total* possible change of the represented thing. By corollary 12 we know that qualitative change occurs within a generalization hierarchy. In other words, in order for qualitative change to occur, there must exist a super- or sub-class. However, since qualitative change involves two distinct state spaces or models, it cannot be described by state transitions (Sec. 5.2.8). Hence, in the case when a super- or sub-class of particular class exists, and qualitative change is possible for a thing represented by an object of said class, an object must possess additional operations that express qualitative change. These operations are not included in those defined to satisfy rule 25. That rule is applicable only to *quantitative* change.

Rule 27 *An object must exhibit additional operations expressing qualitative changes, if a super- or sub-class is defined and instances can undergo changes of class to the super- or sub-class.*

The discussion in Secs. 4.2.3 and 5.2.8 shows that qualitative change is characterized primarily by loss or acquisition of behaviour, represented by operations and state charts. In OCL, this leads us to require that if and only if there exists a super- or sub-class there must exist at least one operation not associated with a state transition expressing re-classification. At most, there exists one such operation for every sub- and super-class. We say "at most" because it may not be possible for a thing to loose properties once it has acquired them.

(5.17)

```
context Class inv:
if self.child->size() + self.parent->size() > 0
then
self.feature
  ->select(f | f.oclIsTypeOf(Operation) and
               f.realization->isEmpty())
  ->size() > 0
and
self.feature
  ->select(f | f.oclIsTypeOf(Operation) and
               f.realization->isEmpty())
  ->size() =< self.child->size() + self.parent->size()
endif
```

5.2.6 Methods

Operations are not the atomic units of behaviour in UML, but are realized by methods. Section 2.3 discussed the concept of a lawful transformation as a 'path' in state space between an initial and a final state and noted that there may exist many such lawful transformations for each pair of initial and final states. With the interpretation of UML operations as state transitions, this leads to the mapping of methods to lawful transformations. Hence, a method describes a lawful 'path' through state space. This path begins in the source state of the state transition represented by the operation which the method implements.

Methods themselves can be described by state charts. The UML meta-model allows the modeller to associate state machines with behavioural features, which are either a method or an operation [8]:

Rule 28 *Methods may be described by state charts other than top-level state charts.*

A lawful transformation may or may not be expressible by a series of state

[8] Here and in many other places, the UML meta-model specifies an aggregation association, which is ontologically incorrect. For reasons outlined in Sec. 2.6 above we refrain from changing this.

transitions. If the transformation represents change of a continuous variable, as is often the case, a series of state transitions can only approximate the path in state space. However, if the change is discrete and there does exist a state chart describing the corresponding method, this state chart must follow some rules, developed presently.

We can think of methods as representing lawful transformations which 'implement', i.e. describe in greater detail, a state transition represented by an operation. As such, the beginning and end of the path in state space represented by a method must match the initial and final states of that state transition. We therefore propose rule 23:

Corollary 23 *A state chart describing a method must begin and end with those states in SC_0 which the operation that the method implements is a realization of.*

To state this in OCL, we make use of initial pseudostates and the fact that final states of a state machine are an explicit subclass of states. The corresponding OCL expression to the above rule then reads as follows:

(5.18)

```
context StateMachine inv:
self.context.oclIsTypeOf(method)
implies
self.top.subvertex->select(sv :
            sv.oclIsTypeOf(State)
            and
            sv.incoming.source.oclIsTypeOf(Pseudostate)
            and
            sv.incoming.source.kind="initial")
= self.context.specification.transition.from
   and
self.top.subvertex->select(sv :
            sv.oclIsTypeOf(FinalState))
= self.context.specification.transition.to
```

Moreover, since a method is a more detailed description of an operation, it must be true that it is caused by the same event that causes the operation. This motivates the following corollary:

Corollary 24 *State transitions out of the initial state of a method realizing an operation must be associated with the same event that is associated with the transition in SC_0 which represents that operation.*

(5.19)

```
context Operation inv:
self.method->forall(m : method |
  m.behaviour->exists()
  implies
  m.behaviour.top.internalTransition
      ->select(tr : tr.source.incoming.source
                      .oclIsTypeOf(PseudoState)
                    and
                    tr.source.incoming.source
                      .kind="initial")
      .trigger
  = self.realization.trigger)
endif
```

Since the UML meta-model allows the association of one model element with any state machine, we suggest that this model element is either a class, in which case the state machine describes the top-level behaviour, or a method, in which case the state machine describes the method. If a state machine is not associated with any model element, it describes a composite state contained which defines sub-states.

Corollary 25 *A state chart either expresses the external behaviour of an object (SC_0), a method, a signal reception or is a composite state contained in another state machine.*

(5.20)

```
context StateMachine inv:
if not
self.context->isEmpty()
then
self.context.oclIsTypeOf(class) or
self.context.oclIsTypeOf(method) or
self.context.oclIsTypeOf(reception)
else
self.SubMachineState->isEmpty() = false
```

This specifically precludes the association of operations with state-machines. If an operation is implemented by a method then this method may be associated with a state machine. If the method defines continuous change between two states, it cannot be described by a discrete state machine. In that case, the state transition that is realized by the operation is an adequate description of change and no state machine is necessary. We therefore suggest the following rule and corollary:

Rule 29 *An operation is not directly specified by state machines. Instead, the methods that implement operations are specified by state machines.*

(5.21)

```
context Operation inv:
self.behaviour->size()=0
```

Furthermore, we propose that state machines which express the behaviour of objects or of methods must not be contained is composite substates in other state machines. The following corollary and OCL expression ensures this:

Corollary 26 *A state machine that specifies the behaviour of a class or a method is not contained in other state machines.*

(5.22)

```
context StateMachine inv:
self.context.isEmpty() = false
implies
self.SubmachineState
   ->forall(sms : SubmachineState |
sms.Container->size() = 0)
```

Rule 28 and corollaries 23, 24 allow the modeller to identify incomplete or redundant state charts and methods: If a method is not associated with a state chart, it may be redundant or it represents a form of continuous change not expressible using discrete change concepts. If a state chart describing a method does not begin and end with the proper states, it may be incomplete.

The BWW-ontology requires that each state transition must change the value of at least one attribute (see Sec. 2.2). This motivates the following rules:

Corollary 27 *The method corresponding to a state chart must modify the attribute values of the object corresponding to the values defined for the initial and final state of the method.*

The adoption of this rule will help to design and specify methods that conform to the state description by ensuring the modeller or designer is provided with exact specifications of the pre- and post-conditions of any method. With the help of the meta-model we can determine the attribute values before and after the execution of a method.

(5.23)

```
context Method:
let preAttributes
  = self.specification.transition.source.slot.value
let postAttributes
  = self.specification.transition.target.slot.value
```

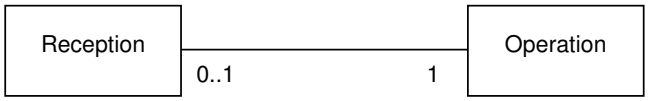


Figure 5.10: Meta model linking signal reception to operations

5.2.7 Signal Reception

This section examines the UML reception construct in terms of ontological concepts. UML provides the construct of a signal reception to specify which UML-signals objects of a certain class can receive and react to. A reception is a behavioural feature like operations and methods.

All behavioural description in UML must be mapped to states and state transitions of the BWW-ontology. UML behaviour already mapped include state charts as well as operations and methods. Receptions form yet another way to express externally induced behaviour and must also be mapped to state transitions. However, receptions do not provide anything that could not be expressed before. Both operations and state transitions can be associated with events which trigger them. For these reasons, the reception construct is redundant in UML.

If receptions are being used by the modeller, there must exist modelling rules which ensure consistency in their use. We suggest that every signal reception correspond to exactly one operation. This is shown in the proposed addition to the meta-model in Fig. 5.10 and made explicit in the following rule:

Rule 30 *An operation must be associated with the declaration of signal reception.*

Since a signal reception and the operation specify the same ontological behaviour, they must be triggered by the same external event. We therefore propose:

Rule 31 *The event associated with an operation must be identical to the event associated with the signal associated with the reception.*

(5.24)

```
context Reception inv:
self.operation.transition->forall(t : transition |
     t.trigger = self.signal.occurrence)
```

Moreover, an operation is specified or implemented by a method. The method in turn may be associated with a state machine which provides details about it. On the other hand, a reception can also be associated with a state machine. Operation and signal reception specify the same *abstract* behaviour, i.e. both specify the same initial and final state, but not necessarily the same path in state space. Thus, instead of requiring that an operation and a reception be associated with the same state machine, we make a weaker requirement:

Corollary 28 *The state machines associated with a reception and with a method specifying the implementation of an operation which is in turn associated with that reception, must possess the same initial and final states.*

We can formalize this using OCL expressions (B.18) and (B.19) in Appendix B to propose the following invariant on signal receptions:

(5.25)

```
context Reception inv:
self.signal.event->forall(e : Event |
  e.transition->forall(t : transition |
     self.behaviour.allInitialStates()
             ->includes(t.source)
     and
     self.behaviour.allFinalStates()
             ->includes(t.target)))
```

Methods, operations and declarations of signal reception are behavioural features. All of them may be associated with a state machine that describes their behaviour. Additionally, an operation is implemented by a method and a signal may also trigger a state transition. Fig. 5.11 shows the relationships among these constructs. Thus, UML provides three different ways to express dynamic behaviour, while the BWW-ontology provides only one way.

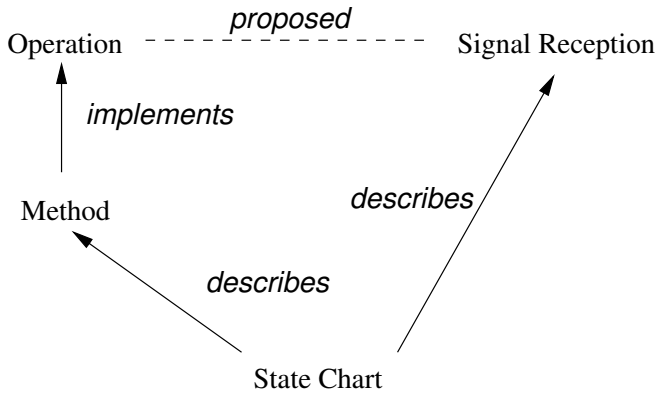


Figure 5.11: Relationships between behavioural constructs in UML

The rules developed in this section give the modeller some guidelines in how to design operations and methods that correspond to behaviour specified in state machines or state charts. Thus, they help ensure correct system specification and system behaviour.

5.2.8 Specialization and Changes of Class

A major problem with specialization is the question of how behaviour, i.e. state machines are specialized. The fact that every instance of the special class is also an instance of the super-class implies that any instance of a subclass may be behaviourally substituted for an instance of the super-class. It must exhibit the same behaviour as an instance of the super-class. What does 'the same behaviour' mean in this context? Prior research (Lakos and Lewis, 2000; Harel and Kupferman, 2000; Schrefl and Stumptner, 2002) has examined this question and proposed rules or guidelines to this effect.

However, none of the past studies considered the integration of behaviour with static structure as exhibited in the BWW-ontology through the definition of states as vectors of attribute values. This integration allows us to identify two criteria for behaviour specialization. The first criterion focuses on substitutability of a sub-class instance with respect to a single state transition:

1. State definitions in terms of attribute values may be interpreted as pre- and post-conditions for state transitions. Hence, state transitions

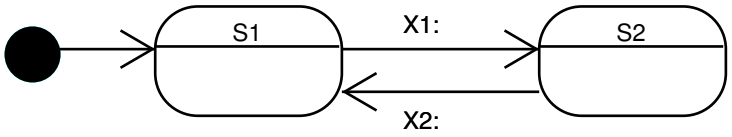


Figure 5.12: Behaviour specialization: States of A

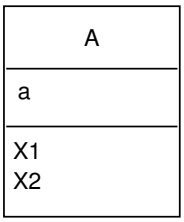


Figure 5.13: Behaviour specialization: Class definition A

in derived sub-classes must conform to these conditions.

Condition (1) is necessary but not sufficient for proper specialization, since it does not guarantee substitutability for *sequences* of state transitions. We therefore add the following condition (2):

2. Behavioural substitutability means that a derived sub-class can undergo the same *sequences* of top-level state transitions as the base class. Equivalently, the derived sub-class can undergo the same series of operations than the base class.

Consider a class A defined with attribute a and two operations, x_1 and x_2. Let two states be defined for instances of A: $s_1 =< a = 1 >$ and $s_2 =< a = 2 >$. This is shown in Fig. 5.12 and Fig. 5.13 respectively. The two operations are associated with state transitions between s_1 and s_2. The state definitions serve as pre- and post-conditions for the operations.

Consider now a specialization of class A by sub-class B, defined in Fig. 5.14. The specialized behaviour, i.e. the top-level state chart of B must conform to the two criteria outlined above. Figs. 5.15, 5.16 and 5.17 show some state charts that fulfill these criteria. In order to simplify the diagrams, we have named the states so as to show their definition in terms of attribute values.

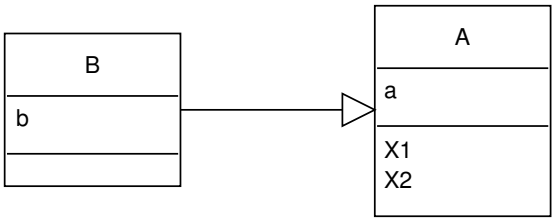


Figure 5.14: Behaviour specialization: Class definition B

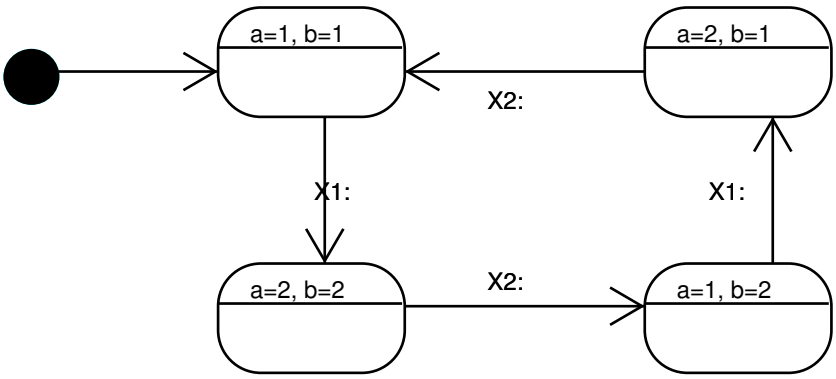


Figure 5.15: Behaviour specialization: Possible state chart of B

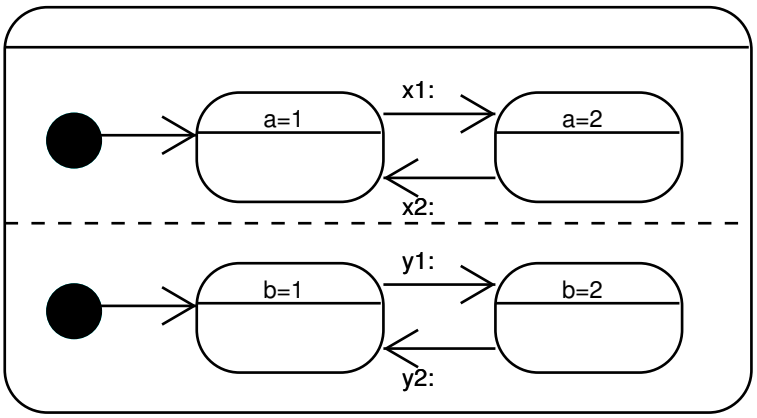


Figure 5.16: Behaviour specialization: Possible state chart of B

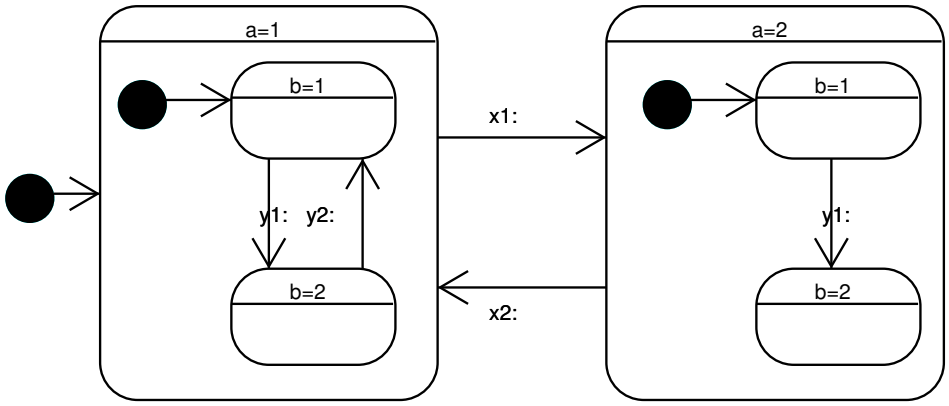


Figure 5.17: Behaviour specialization: Possible state chart of B

Note that in the state chart in Fig. 5.15 the methods $x1$ and $x2$ are each assigned to two state transitions. Depending on the value of attribute b, these methods have different effects. This may be expressed in the method description e.g. through guard conditions and may require overloading the inherited method in an IS implementation. On the other hand, the method may only modify the values of attribute a.

The diagrams in Fig. 5.16 and 5.17 introduce new state transitions which must be realized by an operation. Hence, two new operations, $y1$ and $y2$ are required in the definition of class B which are not defined in the model of Fig. 5.14.

As a more concrete example, consider a customer whose relevant states are defined in terms of an attribute 'OrderVolume'. Assume two relevant states A and B, defined by '$OrderVolume <$ \$100' and '$OrderVolume \geq$ \$100' respectively. Further assume a possible state transition between these states. The two state definitions are now invariant constraints which must be satisfied by all customers that undergo this state transition.

Now assume a special kind of customer, e.g. 'VIP Customer'. In order to be behaviourally substitutable, a 'VIP Customer' must be able to undergo the same state transitions, i.e. changes in the value of 'OrderVolume' must satisfy the same conditions. However, assume that the 'VIP Customer' possesses another attribute expressing the 'Volume Discount Rate' and let this be dependent on the 'OrderVolume'. Then a state transition from a state C defined by '$OrderVolume <$ \$100' and 'Volume Discount Rate = 10%'

to state D defined by '$OrderVolume < \$200$' and 'Volume Discount Rate $= 20\%$' is considered behaviourally substitutable as it does not violate the constraints given by the state definitions. Beyond this *redefined* transition, the 'VIP Customer' may of course exhibit additional behaviour in the form of additional state transition that she can undergo.

Research by Schrefl and Stumptner (2002); LeGrand (1998) argues along similar lines of reasoning. Schrefl and Stumptner (2002) focus primarily on behavioural substitutability in the sense of our criterion (2) above, whereas LeGrand (1998) also considers the fact that states are linked to static structure attributes and their values. The three example state charts shown above conform to results derived in LeGrand (1998). This agreement of an ontological argument with prior research lends further support to both the ontological semantics and the prior research.

Acquisition of Properties The BWW-ontology suggests that additional properties appear as a result of interaction of a thing with other things. This leads to specialization of things in terms of structure (the additional acquired properties) and behaviour (the state changes now possible). There exist two general cases of property acquisition, depending on the relationship of the acquired attributes to the existing ones.

1. Acquisition of independent properties

   Harel (1988) introduces orthogonal regions in state charts as containing concurrent sub-states, i.e. a thing is in two states at the same time. Sec. 5.2.1 shows that concurrent sub-states require mutually disjunct sets of attributes which span the orthogonal regions of the state space. Harel (1988) advocates the use of such orthogonal regions in order to mitigate the problem of exponential growth of the number of states that is experienced in state transition diagrams when state variables are added.

   In the case of independent properties, the additional properties span an independent (orthogonal) subspace of a things state space. Thus, it is possible to employ the construct of orthogonal regions of state charts to support this type of property acquisition. We therefore propose rule 32 which reflects this:

   Rule 32 *Acquisition (loss) of* independent *properties leads to expansion (contraction) of a thing's top-level state space SC_0 by an orthogonal region.*

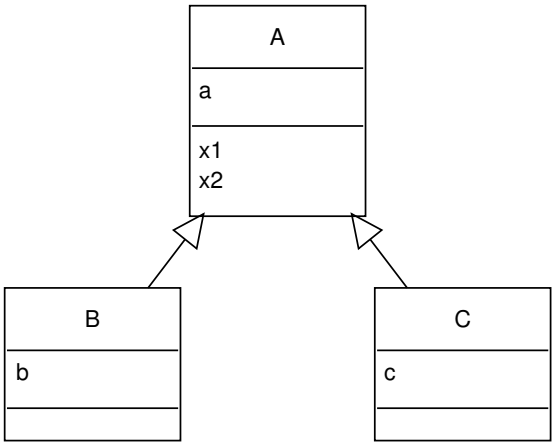


Figure 5.18: Example specialization

If the acquired properties are independent of existing ones, then by corollary 16 (orthogonal regions defined by disjunct attribute sets) it is these and only these that span the new orthogonal region. This agrees with the two criteria regarding behavioural substitutability set forth above: Sub-classes that exhibit independent additional properties are behaviourally substitutable, as the new behaviour is independent of the existing behaviour.

2. Acquisition of non-independent properties

   In the case of acquisition of new properties that are not independent, the existing state-space is redefined and generally the entire state machine description of the object must be re-developed ab initio.

   This may not be necessary in certain cases and it may be possible to develop the derived state machine based on the state machine description for the more general class.

   Consider the situation depicted in Fig. 5.18 showing a general class A with attribute a and two operations x_1 and x_2. There exist two specialized sub-classes B and C which inherit attribute a and define additional attributes b and c. Assume that the domain of a is the set $\{B, C\}$ and that for instances of B, $a = B$ and for instances of C, $a = C$. The value of attribute a then determines not only the state of the instance, but also the sub-class of the instance.

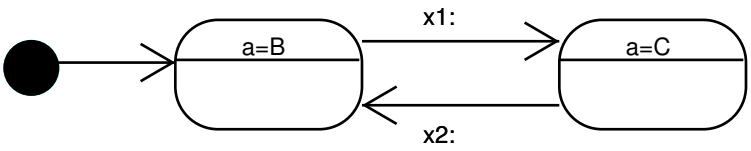


Figure 5.19: Example specialization: State chart

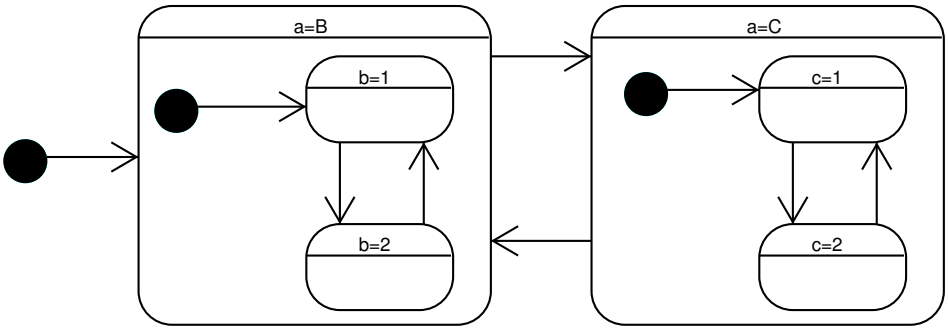


Figure 5.20: Example specialization: Derived state chart

Assume that the state description of instances of A is as shown in Fig. 5.19. Then, according to the above criteria for substitutability, only a state machine description like the one in Fig. 5.20 will be valid. Notice that the attribute b is only present when $a = B$. This in turn implies that the state chart of instances of B will be a sub-state machine contained in state $a = B$. Similarly, the state machine for instances of C will be a sub-state machine contained in $a = C$. Thus, *in the case where a change in an attribute (or a change in state) is also a change in sub-class*, the state machine descriptions of the sub-classes need not be completely be re-defined but are merely a sub-state machine within a more general state description.

An example for this kind of specialization is that of a set of students attending a university. On the level of the students, there exists a property 'ProgramEnrolledIn' which can take on the values 'Undergrad', 'Grad', etc. Depending on the value of the attribute representing this property, the students can be sub-classed into the classes 'Undergrad-Student', 'GradStudent' etc. which in turn possess different attributes

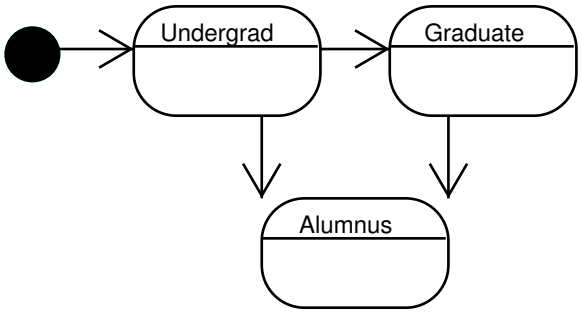


Figure 5.21: Example specialization: Student state charts

such as 'Major' and 'Minor' for undergraduate students, 'NameOf-Supervisor' for graduate students etc. while all students may possess properties such as 'TuitionBalance', 'CreditsAccumulated', etc. In this case, the properties 'Major' and 'Minor' are only present in instances for which 'ProgramEnrolledIn' is equal to 'Undergrad'. Thus, states that depend on the values of 'Major' and 'Minor' are only defined for undergraduate students and can therefore only be substates of student states for which 'ProgramEnrolledIn=Undergrad'.

For practical modelling we suggest that the state chart for each sub-class includes the relevant states of the general class and show the state refinement for that particular sub-class of instances. Figs. 5.21, 5.22 demonstrates this for the student example.

With this discussion of qualitative change, we are now in a position to further formalize the conditions of changeability imposed on UML by the mapping of BWW concepts. The BWW-ontology requires that every thing must be able to change. This implies that either it can undergo a quantitative change or a qualitative change. In order to undergo qualitative change, by rule 12 which states that qualitative change occurs within generalization hierarchies, a class must either possess a more general super- or specializing sub-class. This motivates the following rule, expressing this:

Corollary 29 *Every object must be capable of at least one state transition or be able to undergo change of class to a super- or sub-class.*

With the help of expression (B.13) in Appendix B we can state this requirement in OCL:

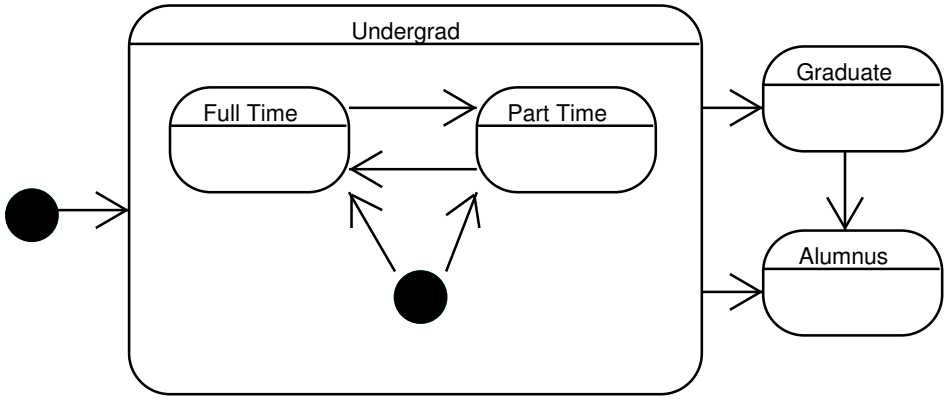


Figure 5.22: Example specialization: Undergraduate student state charts

(5.26)

```
context Class inv:
self.child->size() > 0 or
self.parent->size() > 0 or
self.behavior
  ->exists(sm : StateMachine | sm->OneTransition())
```

5.3 Summary

This chapter examined the notion of change both from the ontological perspective (representation mapping) and, with that mapping in mind, from the perspective of UML (interpretation mapping). We have mapped BWW-states and BWW-state transitions directly to corresponding UML constructs. Other UML constructs have required a more elaborate interpretation. Table 5.1 shows the ontological interpretations of the concepts and constructs discussed in this section.

The table shows that the basic ontological concepts have direct UML equivalents and this mapping relation is again made bijective by the proposed rules constraining the use of the UML constructs. We have noted that the description of change in UML through the use of methods and, at the

Ontological Concept	**UML Construct**	**Remarks**
State	State	Connection with attribute made
State	State Machine	UML allows nested sub-states
State	Sub-state	dto.
State	Action State	Action states are super-states
Transition	Transition	
Transition	Operation	
Lawful transformation	Method	
Law	Guard condition	
	Partition	
	Reception	

Table 5.1: Summary of Interpretations Related to Change

same time the use of state charts, may be ontologically redundant as it must be mapped to the same ontological concepts. This allowed us to propose some rules to keep these two concepts of change consistent. Similarly, the discussion of signal receptions has shown that these may be ontologically redundant. Again, if used, the modeller must obey rules to ensure the model's overall consistency.

Chapter 6

Interaction

In previous chapters we have discussed static structure (Chap. 4) and change within a thing or object (Chap. 5). This section is now concerned with the interaction of two or more things. We will discuss this first from the ontological perspective and then from the UML point of view.

6.1 Representation Mapping

The BWW-ontology does not provide constructs or concepts specifically for the description of interaction. Instead, interaction is defined through state transitions which have already been discussed in Chapter 5. Two or more objects are said to interact, if and only if the state transitions that each object undergoes depend on the existence of the other object. The possible states and transitions are constrained by laws and hence, interactions can in theory be derived from the knowledge of the laws[1].

6.1.1 Interaction & Laws

This section will examine how interactions are caused by laws and how, depending on the laws, a particular state transition may give rise to interaction, i.e. be mapped to a UML-event or UML-action.

[1] We say 'in theory' because for practical purposes this requires methods such as constraint solving (Mackworth, 1977) or constraint programming (Van Hentenryck, 1989) which are not generally applicable and, except for special cases, not computationally efficient, i.e. not of polynomial computational complexity.

Ontologically, interaction is *described* by state histories. When the state history of one thing depends on the existence of another, there exists interaction. This is *not* a mechanism of interaction. Interaction arises because things adhere to laws that must be satisfied at all times: When an event in thing A changes a property which is lawfully related to a property of thing B, the property of thing B must adjust so that the law remains satisfied[2]. Thus, interaction is a primary concept in UML and the object approach (in the sense that it is directly specified, not derived), it is derivative in ontology: Interaction can be deduced from properties and the laws that relate them to one another.

Object-oriented approaches express interaction through message-passing. However, any given message passing pattern is simply one specific way of satisfying the laws and there may be

1. other mechanisms beside messages passing or
2. other patterns of messages that also satisfy the laws.

The ontological nature of message passing is discussed in Sec. 6.2.1 below as part of the interpretation mapping.

In general, it is the system designer's task to ensure that the modelled message-passing pattern satisfies all laws. Hence, message-passing and the modelling of message-passing is a design oriented task. It is often the case that such laws are not explicitly stated, but only implicitly understood by the analysts. Our ontological evaluation can help by explicating the existence of laws or constraints and suggesting possible ways to use message-passing to satisfy the laws. In sum, the BWW ontology supports a descriptive and declarative style of modelling whereas the object-oriented approach is more prescriptive and procedural. This is not surprising as early object-oriented approaches were rooted in procedural programming. Early object-oriented programming systems were pre-processors that would translate object language code, e.g. C++, into corresponding procedural code, e.g. in C. Methods were realized by implementing pointers to function in the same structure as variables (attributes of objects).

We use the remainder of this section to examine when and how interaction should be modelled. We do this before the background of the message

[2]Note also that an information system (or any artificial system) is *always* in a *lawful* state. However, this state may be *undesirable* to the IS designer and so is a symptom of a design fault.

passing paradigm that is prevalent in object-oriented approaches and UML. Specific UML constructs will be discussed in the following interpretation mapping.

In the discussion of the BWW-ontology in section 2.3 we remarked that since laws constrain the state variables of only a single thing, changes among two things must happen by virtue of mutual properties. If the two things are parts of the same composite, changes among them may happen by virtue of emergent properties. Thus, a change in one thing not only leads to a change in another thing but *is* a change in another thing. Only one property is changed by a state transition in one thing and this change is a change of the *same* property in the other thing and thus *is* a state transition in the other thing.

With the mapping of mutual properties to association class attributes, this discussion leads us to propose the following rule for modelling interaction in UML:

Rule 33 *For every two classes of objects between which message passing is declared, there exists an association class or the two classes are parts of the same aggregate.*

In OCL, making use of expressions (B.6) and (B.7) defined in Appendix B:

(6.1)

```
context Message inv:
self.sender.base.allProperties()
   ->exists(p : attribute |
                p.owner.oclIsTypeOf(AssociationClass)
                and
                self.receiver.base
                    .allProperties()->includes(p))
or
self.sender.base.partOf()->intersect(
  self.receiver.base.partOf())
->isEmpty() = false
```

This rule is helpful for the modeller to identify which messages need to be shown in a model and between which two object an association class needs to be identified.

There are some situations when this rule may need to be relaxed in practice. It is not always possible nor desirable to model intermediate objects such as communication systems. Instead, these are often taken for granted as they are outside the scope of the system. For example, a telephone line between two people possesses mutual properties with both the speaker and the listener. It is these mutual properties and the properties of the phone line (voltage, waveform, etc.) which undergo change. In practice, the telephone line and it's properties will not be included in the model and instead be assumed implicitly.

Bunge (1977) proposes that every thing acts on and is acted upon by other things. This means that there must exist some interaction originating and terminating in any particular thing:

Rule 34 *Every object must be the receiver and sender of some message.*

While this is a very strong requirement, it shows the need to critically examine the interactions or information flows. One could argue that things in a system's environment may only send but not receive messages, e.g. a customer ordering a product, a student registering for a course, etc. But even for these examples, the actions of ordering and registering are pointless unless the customer and the student also receive messages. This could be order acknowledgements or shipping details for the customer. For the student, the messages may involve tuition fee billing or text book requirement messages. In other words, neither the customer nor the student will only send messages without receiving any. Note also that messages in UML are more general than information, but represent the change of mutual properties. Thus, the customer receiving the shipment and the student being invoiced the tuition fees is also represented by messages.

Often, a thing is an aggregation of parts. In such cases, it is sufficient if the aggregate or any of its parts receive and send some message. The receiving and sending must not necessarily be done by the same part. Assume there exists an aggregate a with parts p_1 and p_2. If p_1 receives messages but does not send any and p_2 sends messages but does not receive any, we consider rule 34 to be satisfied, as the aggregate a both receives and sends messages. With this, the above rule 34 can be expressed in OCL as follows, making use of expressions (B.5) and (B.7) defined in Appendix B:

(6.2)

```
context Class inv:
self.allParts()
->union(self.partOf())->exists(cls : Class |
   cls.ClassifierRole->exists(clr : ClassifierRole |
       clr.sender->size() > 0)) and
self.allParts()
->union(self.partOf())->exists(cls : Class |
   cls.ClassifierRole->exists(clr : ClassifierRole |
       clr.receiver->size() > 0 ))
```

This OCL expression specifies that some part of a class or some aggregate of which the class is part of (or their generalized super-classes from which they inherit) plays the role of sender of a message and some other part of a class or some other aggregate of which the class is part of plays the role of receiver of a message.

This implies by rule 33, that there exist mutual properties or emergent properties of common aggregates that relate any one thing to some other thing. Moreover, these must exist before the message is passed. In most practical situations, these are mutual properties of intermediate objects that may be omitted, such as the telephone line in the example above.

As association classes are interpreted as representing bundles of mutual properties, *not* substantial things, they can be part of interaction only by being changed as part of change in the objects of participating classes. They represent the things possessing the mutual properties undergoing that change. Association classes do *not* represent things that are capable of change. Furthermore, rule 4 suggests that behaviour should be modelled with the participating classes. This leads us to propose the following rule:

Corollary 30 *An association class cannot be the sender or receiver of a message.*

Hence, association class instances cannot interact with other instances, reflecting the fact that in ontology mutual properties do not interact, that it is the things possessing them that interact. As an example, consider the mutual property of a salary of an employee with a company. While the employee and company are substantial things that can interact, the salary

is a property that cannot interact, even though it may be changed as part of changes in either the employee or the company (e.g. when the company promotes the employee and raises the salary). In OCL:

(6.3)

```
context Message inv:
not self.sender.base.oclIsKindOf(AssocationClass) and
not self.receiver.base.oclIsKindOf(AssociationClasse)
```

Laws

UML provides the construct of a constraint. A constraint relates two or more elements of a model and restricts their value. We propose to map a constraint to a BWW-law. For this to be a valid mapping, we must restrict UML-constraints to relating attributes of the same object, i.e. declared within the same class. Rule 35 ensures this:

Rule 35 *A constraint relates attributes of a single class or attributes of association classes the class participates in.*

In OCL this is described by the following OCL expression (6.4), making use of expression (B.14):

(6.4)

```
context Constraint inv:
self.constrainedElement->forall(c1, c2 :
  c1 <> c2 and
  c1.oclIsKindOf(Attribute) and
  c2.oclIsKindOf(Attribute) and
  ( c1.propertyOf()->includes(c2.propertyOf()) or
    c2.propertyOf()->includes(c1.propertyOf()) )
```

6.2 Interpretation Mapping

This section examines the UML constructs and concepts that are related to interaction, while keeping in mind that these must be interpreted in terms of state transitions and laws. The central concept of object-oriented interaction is that of message passing. We begin this section by examining the language constructs that UML provides to support this interaction paradigm.

The discussion of the BWW-ontology in section 2.3 showed that all properties are lawfully related to others. We therefore propose the following rule:

Rule 36 *For every attribute there exists a constraint which relates this attribute to some other attribute.*

For example, the throughput of a machine is constrained by its possible input, by the production schedule, by the material of the parts it works on. Similarly, the expected quarterly profit is constrained by economic expectations of the management team, the number and productivity of the group of employees comprising the company and many other factors. In general, there exists no property that can be freely changed without affecting or being affected by constraints that relate it to other properties.

As constraint conditions are not decomposable in the UML meta-model, this rule cannot be expressed in OCL.

6.2.1 Message Passing

Message passing is one of the core concepts of the object-oriented approach. Message passing is the mechanism by which object interaction and object communication is realized. UML supports message passing through a number of constructs. Stimuli (instances of messages) are passed along links (instances of associations) between objects (instances of classes). The stimulus specifies the nature of the communication, i.e. which operation to invoke or which signal to raise. A stimulus is created and sent by an action. Receipt of a stimulus is an event. Different actions may create different stimuli. The UML meta-model does not explicitly sub-class the stimulus construct. The class of stimulus is determined by the associated action that created it:

- Call action – Synchronous or asynchronous method invocation
- Send action – Results in the (asynchronous) sending of a signal

- Create action – Creation of an instance of some class
- Destroy action – Destruction of the target object
- Return action – Returning a value to the sender of the call action
- Terminate action – Results in the self-destruction of an object
- Uninterpreted action – Any other action which has no interpretation in the UML meta-model

The first four of these actions create a stimulus for inter-object communication. While the create action does generate a stimulus, UML does not associate a target object with a create action. The last three do not create a stimulus. In the UML meta-model a stimulus is associated with the action that dispatches it. Hence, there is no need to explicitly associate a stimulus with a signal (of which it may be a carrier) or an operation (whose invocation it may signify), as the sub-class of the dispatching action determines what effect the stimulus will have. Since objects, with the ontological interpretation of things, cannot be destroyed or created, there is no creating or destroying thing that a stimulus could be sent to. Hence, only the first two of these actions are relevant to the interpretation of message-passing.

Stimuli can be interpreted in two ways, depending on the ontological status we ascribe to them:

- Stimuli are not things in the world. They are abstract concepts that serve as descriptions, illustrations, abstractions or representations of interaction.
- Stimuli are substantive things in the world. They are ontologically real.

There are two main arguments against the second interpretation. First, consider messages or stimuli such as:

- The machine sends a message to a part to change its location.
- The general ledger sends a message to an office desk to depreciate its value.
- A truck sends a message to the crate asking it to load itself onto the loading dock.

Such messages among objects have not been observed in the real world. Instead, we find that machines interact with parts to change their location, rather than sending a message. General ledgers do not send messages, but companies depreciate the value property of inventory and forklift operators interact with forklifts and crates and thus change the latter's location. No messages are sent in this case either. In special circumstances, we may see message passing in the real world, e.g. between two human actors or between interconnected machines or information systems. This is message passing in a much more special sense than the message passing in the object paradigm, which is used to describe any kind of interaction. This argument confirms indications that messages may be a construct related more to IS design than the real world, as suggested by Wand (1989); Parsons and Wand (1991); Wand and Weber (1993).

Second, if stimuli were real things in the world, then instead of two objects interacting directly, one object would have to interact with a stimulus, which in turn would need to interact with the second object. This only defers the problem to the interaction of a stimulus thing with another thing instead of the two original things.

This discussion showed two reasons against interpreting stimuli as ontologically real and we therefore consider stimuli as conceptual abstractions of interaction, not as things.

6.2.2 Stimuli, Actions & Events

With the ontological status of message-passing in mind, the following paragraphs examine various UML constructs related to message-passing. We begin the discussion by interpreting stimuli, actions and events before proceeding to the special cases related to signaling or method calling.

Stimulus

A stimulus is an instance of a message that is sent between two objects, i.e. it has a sender and receiver. It is dispatched by an action and travels along a link. It forms the mechanism of interaction.

The BWW ontology does not specify any mechanism for interaction, although all laws must be satisfied at all times[3]. Given the above discussion,

[3]While there exist techniques in IS design and implementation that allow the same (e.g.

we propose that stimuli have no equivalent in ontology and hence are ontologically excessive. Since a stimulus is not ontologically real, there is also no ontological equivalent to a message, the specification of a stimulus.

Actions

A UML-action is the creation and dispatch of a stimulus. Since the previous paragraphs interpret stimuli not as ontologically real but as an abstract description of interaction, a UML-action is similarly interpreted as change in one object that brings about a change in another object. In the BWW-ontology this kind of interaction occurs when a BWW-state transition in one thing changes a mutual property of two things, thus also being a change in the other thing, since mutual properties span the state space of both[4].

Hence, we map a UML-action to the ontological concept of a BWW-state transition, specifically a state transition in the acting thing. With the interpretation of mutual properties as association classes (Sec. 4 we can then formulate the following rule, which is limited to *quantitative* change:

Corollary 31 *A UML-state transition associated with an action must modify an association class attribute's value or an emergent property of an aggregate.*

With the help of OCL expressions (B.16) and (B.17) in Appendix B we can define the following OCL expression to formalize rule 31:

(6.5)

```
context Transition inv:
self.action->exists() and not
self.isTopLevel()
implies
self.changesMutualProperties() or
self.changesEmergentProperties()
```

constraint programming, constraint solving techniques (e.g. Mackworth, 1977; Van Hentenryck, 1989)) in most object-oriented techniques the designer is responsible for specifying a suitable mechanism for ensuring the satisfaction of all laws at all times.

[4]See also Sec. 6.1.1

We restrict this rule to UML-state transitions in method descriptions, i.e. we exclude those in top-level state charts, because operations may leave the object in the same state or in a state in which at least the mutual or emergent properties possess the same values. This however cannot be the case in the more detailed decomposition of such a top-level state transition in a method.

Events

Section 5.2.5 mapped UML-events to the ontological concept of a state transition. Given that the dispatching UML-action of a stimulus is mapped to a state transition in the acting thing, we propose that a UML-event signifying reception of the stimulus be mapped to a BWW-state transition in the thing acted upon. It is that state transition in the acted upon object which corresponds to the changes in the mutual property caused by the corresponding UML-action.

With this interpretation, the following corollary relates interactions to the state charts of the interacting objects:

Corollary 32 *For every interaction between UML-objects, there must exist a corresponding UML-state transition in both interacting UML-objects.*

Like corollary 31, corollary 32 is applicable to *quantitative change only.* Separating the interaction into action and event, we can formulate the following rules in OCL:

(6.6)

```
context Action inv:
self.transition->size()=1
```

(6.7)

```
context Event inv:
self.transition->size()=1
```

This rule helps the modeller with the identification of state transitions and methods, since the latter are related to top-level state transitions by

way of the operations which they implement. From the observation of interaction, the modeller can thus derive an initial list of state transitions and methods of objects.

Analogous to the justification for corollary 31 we propose that every transition which is externally induced, i.e. associated with an event, must modify the values of some mutual or emergent properties, the former represented as association class attributes.

Corollary 33 *A state transition associated with an event must modify an association class attribute's value.*

(6.8)

```
context Transition inv:
self.event->exists() and not
self.isTopLevel()
implies
self.changesMutualProperties() or
self.changesEmergentProperties()
```

Similar to the argument for rule 31 we also restrict the scope of this rule to transitions not part of the top-level description of object behaviour.

With this interpretation a UML-event is distinct from the ontological notion of an event. In the BWW-ontology, any state transition is an event. In contrast, a UML-event corresponds to an ontological event iff the state transition was externally induced, i.e. resulting two things being related by law or a mutual property. The proposed modelling rules in this and the previous chapter support this distinction.

Having interpreted the general case of actions and events, we now proceed to investigate the two special cases of object interaction, sending signals and invoking operations.

6.2.3 Signal Events and Send Actions

For the same reasons for which we interpret a stimulus as not ontologically existent, we decide against interpreting a signal, carried by a stimulus, as ontologically real. Instead, it too, serves as an abstraction of object interaction

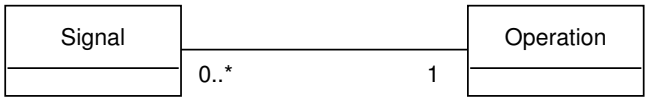


Figure 6.1: Signals are associated with operations

which we have mapped to state transitions. Together with the discussion in Sec. 6.2.2 this leads us to conclude that it is ontologically excessive.

Signal events, as events, may be associated with state transitions in the receiving object. In Sec. 5.2.5 we proposed that state transitions that are associated with an event must be realized by an operation. Since they originate by definition from a stable state, they are top-level transitions. We propose to make this relationship explicit through the additional association in the UML meta-model shown in Fig. 6.1. In addition we require the following OCL invariant to maintain consistency. It states that for every signal, all associated signal events are associated with transitions that are realizations of the operation which the signal is associated with.

(6.9)

```
context Signal inv:
self.signalEvent->forall(se : SignalEvent |
       se.transition.realization=self.operation)
```

This interpretation of a signal event implies that signal events can only be associated with state transitions in top-level state charts (state charts that are associated with classes), as those state transitions correspond to operations being performed (executed). This leads to the following rule:

Corollary 34 *A signal event may only be associated with a transition in a top-level state chart and the initial transition of a method implementing this.*

In OCL expression (6.10) we ensure that either there exists a combination of two associated transitions satisfying rule 34 or, in case no method implementation is given in terms of a state chart, the signal is only assigned to top-level transitions.

(6.10)

```
context SignalEvent inv:
self.transition->exists(t1, t2 |
  t1.isTopLevel() and
  t2.source=t1.source)
or
self.transition.isTopLevel()
```

Next we look at the second mode of inter-object communication, that of method calling.

6.2.4 Call Events and Call Actions

In Sec. 5.2.5 we have mapped operations to state transitions in top-level state charts. Similar to our arguments for stimuli and signals we propose that the call action corresponds to a state transition in the calling thing, while the call event is mapped to a state transition in the called thing (thus rendering either the operation or the call event ontologically redundant). Analogous to the rule restricting signal events to top-level transitions, we define the same restriction for call events:

Corollary 35 *A call event may only be associated with a transition in a top-level state chart or the initial transition of a method implementing this.*

The corresponding OCL expression (6.11) is analogous to expression (6.10):

(6.11)

```
context CallEvent inv:
self.transition->exists(t1, t2 |
  t1.isTopLevel() and
  t2.source=t1.source)
or
self.transition.isTopLevel()
```

In the UML meta-model, the call action and call event are associated with the operation that is invoked. Because top-level state transitions are also associated with operations, the operation specified for the call action and call event must be the same as the operation specified as the realization of the state transition that is triggered by the call event:

(6.12)

context `CallEvent` **inv:**
`self.transition.realization=self.operation`

6.2.5 Get and Set Messages

Object-oriented models often include 'Get' and 'Set' messages specifying corresponding stimuli and invoking corresponding operations. These are a direct consequence of the idea of encapsulation which is central to the object-oriented approach: No other thing can set or get a value of an attribute that belongs to a particular thing but through such messages. The purpose of such messages is thus for the object to either influence or be influenced by some other object's state. Hence, Get and Set methods or messages specify a mechanism for interaction.

In the BWW-ontology, the principle of encapsulation does not hold. Mutual properties, by which interaction occurs, are exactly that: mutual. They are shared properties of two or more things. Thus, interaction can be described without artificially separating the interacting things. 'Get' and 'Set' messages are an implementation driven constructs.

'Set' operations may be interpreted as any other operation, corresponding to a top-level state transition. Like all other state changes, they must be a result of the laws governing the things and reflected in state charts.

The 'Get' operations are ontologically excessive: If an object A sends a 'Get' stimulus to object B, it means that some state information about object B is or should be shared with object A. In contrast, ontologically, interaction is described through mutual properties spanning state spaces of two or more things, and therefore the explicit exchange of state information is not necessary. Rather, to conform with the BWW-ontology, such state information can only be information about mutual properties. Hence, the modeller must identify these mutual properties and model them as associa-

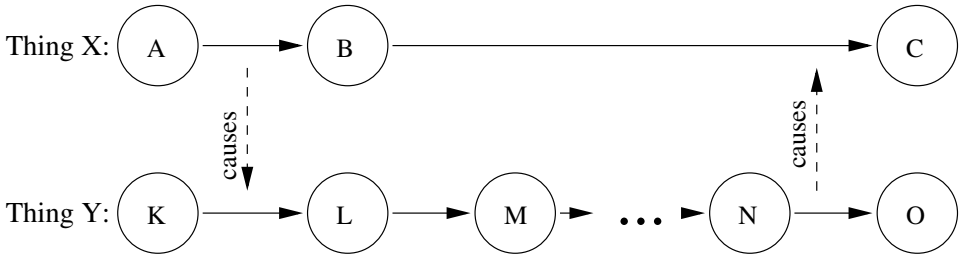


Figure 6.2: Synchronous communication

tion class attributes.

Instead of a 'Get' operation, the model must properly express interaction so that the association class attributes which represent the mutual properties, are updated to reflect the current state. In other words, a 'Get' message indicates that the modeller missed some mutual properties which changed during interaction that occurred prior to the Get operation.

6.2.6 Synchronous and Asynchronous Communication

Object interaction can be synchronous, in which case the acting object waits for a response from the object acted upon, or asynchronous, in which case the acting object does not wait for a response or a response is not required. Ontologically, this difference can be described using appropriate sets of allowable state transitions.

Suppose an object X communicates synchronously with an object Y and Y returns a result. This is interpreted ontologically as thing X causing a change in Y, thereby putting thing Y into an unstable state and starting an internal process. While thing Y goes through the series of unstable states, thing X is in a stable state that can only be changed by changing a mutual property of X and Y by Y: X acts on Y and then Y acts on X to relay the result. An example of this is shown in Fig. 6.2 where the first object transitions from state A to state B, causing a state transition in the second object from K to L, thus beginning an internal process. At the end of that process, the state transition from N to O causes the first object to transition out of stable state B to state C, indicating some result. This discussion leads us to propose the following rule:

Corollary 36 *Synchronous communication of objects implies transition to a state which cannot be left except through a state transition associated with the return signal.*

This rule allows the modeller to keep state diagrams and interaction diagrams such as sequence diagrams consistent with one another.

When the object communicates asynchronously with the other object, the restrictions of corollary 36 on the occurrence of other transitions out of B are not applicable. We can distinguish two cases:

- Asynchronous communication without reply
- Asynchronous communication with reply

The first case is trivial. Since no reply is necessary or expected by the acting object, no additional restrictions are placed on the actions of either object.

The second case implies that the acting object must eventually enter a state in which the response from the object acted upon can be received. In other words, at least one state that the acting thing can enter after transition from B must allow for at least one state transition caused by the last transition in the object acted upon, i.e. that from state N to state O, indicating the response. This leads us to propose the following rule:

Corollary 37 *Asynchronous communication of objects with expected response implies the existence of at least one state transition caused by the object acted upon, signifying the return interaction after the state transition signifying the original communication.*

Communication by signals is always asynchronous (OMG, 2001) and no result is implied. Hence corollaries 36 and 37 do not apply and no further restrictions are placed on the states and state transitions of either object.

Communication by method invocation through call events and call actions may be either asynchronous or synchronous, in both cases requiring a response interaction[5]. Hence, depending on the call action, either of the above corollaries 36 and 37 apply. Since UML provides the construct of a "Return action" without an associated "Return event", the formalization of

[5]If no response is required or expected, send actions using signals should be employed.

corollary 36 in OCL is problematic. However, we can ensure that every final state transition of every state chart describing every method implementation of an operation called by a call action is associated with a return action:

Corollary 38 *The final state transitions of any method implementing an operation that may be invoked through a call action must cause a return action.*

This is ensured formally in OCL by the following expression:

(6.13)

```
context Transition inv:
self.effect.oclIsTypeOf(CallAction) and
self.effect.isAsynchronous=false
   implies
self.effect.operation.method->forall(m |
   m.behaviour.top.internalTransition->forall(t |
       t.target.oclKindOf(FinalState)
       implies
       t.effect->size()=1
       and
       t.effect.oclIsKindOf(ReturnAction)))
)
```

In OCL, we can ensure that if a transition causes an asynchronous call action then there exists another transition within that super-state which is caused by an event resulting from an action by the target of the original send action:

(6.14)

```
context Transition inv:
self.effect.oclIsTypeOf(CallAction) and
self.effect.isAsynchronous=true
   implies
self.state.internalTransition->exists(tr : transition |
   tr <> self and
   tr.event->size()=1 and
   tr.event.signal.sendAction->exists(sa : CallAction |
       sa.target=self.Machine().context))
```

With this ontological interpretation of the two types of object communication, both sending of signals and calling of methods are mapped to the same ontological concepts. Ontologically the main difference is not that one relates to behaviour by methods while the other relates to behaviour by state changes, as is the case in the UML description, but rather that one type requires a response whereas the other does not. In ontological terms, while a send action represents action, a call action represents interaction.

This discussion also shows that for any state transition's effect to be a return action, that particular method must have been invoked first by a call action. This leads us to propose the following rule:

Corollary 39 *For the state machine of a method to contain a state transition whose effect is a return action, there must exist a corresponding state transition in a state machine of some other object whose effect is a corresponding call action.*

We can express this in OCL as an invariant on the return action construct:

(6.15)

```
context ReturnAction inv:
self.transition.Machine().context
    .specification.callAction->Size() > 0
and
self.transition.Machine().context
    .specification.callAction->exists(ca |
        ca.target =
        self.transition.Machine().context.owner)
```

6.3 Summary

Earlier work by Wand (1989); Parsons and Wand (1991); Wand and Weber (1993) argues that messages are an implementation related construct and may be ill-suited for conceptual modelling of the real-world. Our interpretation of stimuli and signals as ontologically not existent confirms their conclusions. In the real world, object interaction is a result of laws that must be satisfied, no mechanism is needed. Message-passing is an abstraction that may serve purposes of software design and implementation or as a metaphor for illustrating certain software mechanisms. The modeller becomes responsible for finding 'correct' patterns or sequences of interaction.

To summarize, in this section we have examined interaction from the perspective of the message-passing paradigm prevailing in object-oriented approaches. We have examined the UML constructs provided to support this paradigm and have interpreted them in terms of BWW-state transitions and laws. Table 6.1 shows the ontological mappings that we have made.

This summary table shows that while the message-passing paradigm and the associated constructs are related to IS design and implementation, most of them can be interpreted in terms of the standard BWW-concepts related to interaction. When these mappings are made, it leads to construct redundancy and care must be taken to ensure consistency. The proposed rules can help the modeller with this task.

Ontological Concept	**UML Construct**	**Remarks**
	Stimulus	Not ontologically real
	Message	dto.
	Signal	dto.
Transition	Event	Of the thing acted on
Transition	Action	Of the acting thing
Transition	Call Event	
Transition	Call Action	
Transition	Signal Event	
Transition	Send Action	
Law	Constraint	
	'Get' message	Indicates prior interaction
Transition	'Set' message	Interpreted like other actions

Table 6.1: Summary of Interpretations Related to Interaction

Chapter 7

The Object Paradigm

In Chapters 1 and 2 we have argued for a careful and cautious process in assigning semantics to UML. The discussion in those chapters recognized the danger of re-defining a language beyond recognition and re-defining it so far that it fails to be the language familiar to IS designers. This is a major concern when re-interpreting any language and proposing rules that alter the admissible construct combinations. The question whether the essence of the language remains must be explored.

In the case of this research we need to examine whether the suggested guidelines and changes to the meta-model maintain the object-oriented characteristics and features of the language. In other words: Have our changes redefined UML beyond recognition or is it still essentially an object-oriented language?

To answer this question, we examine the commonly accepted fundamental principles that define the object-oriented paradigm. While there is no unique list of generally principles of the object-oriented approach to IS development, there is some convergence among researchers. For example, Parsons and Wand (1997) have compiled a set of characteristics and suggest the following principles:

- Identity
- Encapsulation
- Persistence
- Homogeneity
- Classification and Instantiation
- Specialization and Inheritance
- Composition

- Communication and Interaction through message-passing
- Relationships
- Polymorphism
- Dynamic Binding
- Concurrency

where Barker *et al.* (1993) examine the object-oriented paradigm and suggest the following general characteristics:

- Objects and Encapsulation
- Message Passing
- Classification and Instantiation
- Subclassification and Inheritance
- Polymorphism

Korson and McGregor (1990) suggest the following essential characteristics of the object-oriented approach to IS development:

- Objects and Encapsulation
- Classification
- Inheritance
- Polymorphism
- Dynamic Binding

Other feature compilations (Capretz, 2003; Fil, 1999; Kirstensen and Osterbye, 1996; Wegner, 2003) provide similar feature sets as defining for object-oriented modelling languages. Our critical assessment of the proposed changes focuses on the following six principles. These six criteria form the common core of the research referenced above and are generally accepts as the main constituents and principles of the object-oriented paradigm.

Encapsulation This principle suggests that objects consist of structure and behaviour. While the structure part of objects, the attributes, map more or less directly to ontological properties of things, the behavioural aspect, operations and methods require a more complicated mapping.

Our suggested rules do not violate the principle of encapsulation except in one case. We do not allow the modelling of behavioural features for association classes. As association classes serve only as a 'container' for their attributes and do not correspond to substantial things, the corresponding behavioural features must be ascribed to the classes and objects participating in the association. Hence, the attributes of an association class,

representing mutual properties, are accessible by the participating classes.

On the other hand, our approach retains encapsulation in the sense that only mutual properties represented by association class attributes are accessible. Intrinsic attributes of regular classes are by their very definition not publically accessible and therefore adhere to the principle of encapsulation. They are changed only by internal processes.

Classification Objects of similar structure and behaviour form classes. Given that object-oriented modelling languages derive from programming languages, a class is not considered a collection of objects but a specification of objects.

In ontology, we define classes as sets of things with similar properties. Bunge (1977) suggests that laws of a thing are a kind of property, where property is used in a wider sense. As laws determine the behaviour of things, ontological classes and kinds are thus formed by things of similar structure (properties in the narrow sense) and behaviour (properties in the wider sense).

Our rules do not prohibit the formation of classes. On the contrary, the rules suggest what should or should not be modelled as an object and can therefore be classified. UML association classes are one instance where our rules do prohibit the classification of an entity that is classifiable in regular UML. The attributes of association classes correspond to mutual properties which are properties of their respective things, not a third 'intermediate' thing. However, our suggested rules do allow the modelling of a class even in this case, but we place a number of constraints on such an association class.

A slight difference is our focus on behaviour for determining when sub-classification should occur. In UML any additional feature, regardless of whether this is a static or behavioural feature, leads to sub-classification. In our proposal, it is primarily additional behaviour which necessitates sub-classification. However, this additional behaviour is generally concomitant with additional properties.

Inheritance Inheritance allows objects of a sub-class to inherit the features of the superclass. Objects of sub-classes can also possess additional features and re-define inherited behavioural features. Sub-classification and inheritance are orthogonal features. In other words, inheritance may be

used for sub-classification, but need not be employed for this (Meyer, 1996; Tailvalsaari, 1996).

As the semantics of classification is slightly different in ontology than it is in UML and other object-oriented design languages, so are the semantics of sub-classes and therefore inheritance. Classes in UML are merely descriptions of objects, not collections. Therefore, defining a sub-class and using inheritance is a descriptive technique, it operates on the syntactic level to aid reuse of specifications.

On the other hand, the semantics of a class in ontology is that of a set (of things sharing properties or behaviour). Hence, a sub-class is a sub-set of some set. Hence, the elements of the subset quite naturally possess the features they share with the elements of the complement of the sub-set. Therefore, there is no need in ontology for the notion of inheritance, as it is not concerned with the description or specification of things.

The rules that we suggest for UML do not prohibit the kind of inheritance expressed or expressible in UML. On the contrary, it is a helpful tool for describing what exists in a domain. Our rules and semantics are compatible with the notion of inheritance in object-oriented languages. This is evident from the OCL expressions, a number of which make heavy use of inheritance of features and work well within the inheritance hierarchies provided by the meta-model.

Polymorphism Polymorphism is the ability of objects to interact with other objects without being aware of their exact class. In other words, as long as a particular object that is acted on can exhibit a certain behaviour, it is irrelevant to which class this object belongs. In common object-oriented languages, polymorphism is generally restricted to objects of sub-classes of common super-classes.

While the BWW-ontology does not employ message-passing and interaction is a primary, derivative concept, there are analogs to polymorphism in the BWW-ontology. Interaction is described through changes in mutual properties. As such, interaction is not specific to any particular class or kind of thing that exhibits this property. Hence, the BWW-ontology exhibits polymorphism in this particular way. In contrast to object-oriented approaches, this kind of polymorphism is not restricted to sub-classes of a common super-class.

Polymorphism is a natural aspect of the BWW-ontology. This is not

surprising given that the BWW-ontology is primarily instance based, rather than class-based. Things with their properties and behaviour are primary. Classes are defined on top of sets of things. Thus, it is irrelevant for a thing which class it is an element of. What matters in the ontology are the properties and behaviour of a specific thing.

Moreover, we have examined behavioural substitutability which is an important aspect of polymorphism. The two criteria that were developed are in agreement with other research (Schrefl and Stumptner, 2002; LeGrand, 1998) that suggests conditions for polymorphism interpreted as behavioural substitutability. Hence, our ontological semantics does not preclude the notion of polymorphism in interaction and is compatible with the general notion of polymorphism in UML and other object-oriented approaches.

Message-Passing Message-passing is one of the central principles of object-oriented approaches and specifies the interaction mechanism by which objects communicate. As discussed in the previous chapters, the BWW-ontology makes no claim to such a mechanism and we found some of the constructs involved in message-passing to be excessive.

However, we have attempted to interpret message passing as an abstraction of interaction. As a result, we were able to suggest a mapping of some of the language constructs that implement message-passing to ontological concepts. Our suggestion is that message passing be used to describe interaction, but that modellers keep in mind that interaction is described through changes in mutual properties, expressed as changes of association class attributes.

Thus, while use of the message construct is permissible and encouraged e.g. in sequence diagrams or collaboration diagrams to indicate the interaction pattern of objects, this must lead, by virtue of our rules, to the specification of appropriate association classes and state charts in other parts of the model.

Information Hiding/Abstraction As identified in the discussion of encapsulation, the one instance where our rules violate the principle of information hiding is that of association classes. Because of our mapping, attributes of association classes are modified by operations or methods defined for the classes participating in the association. No other rules violate this principle.

To summarize, this examination of six fundamental principles of the object-oriented approach to systems specification shows that our suggested semantics support all of the principles. The only exception to this is the violation of encapsulation for association classes. Encapsulation can be restored by allowing and introducing "Get" and "Set" operations for association class attributes. Thus, if desired for software design and implementation models that must fully conform to the principle of encapsulation, such a simple transformation will have the desired effect. This transformation should be applied only after the conceptual model is fully developed.

In summary, the changes to the meta-model and the rules will lead to valid object-oriented models. UML, with the proposed semantics and with the proposed rules, remains the object-oriented language that is familiar to software designers.

Chapter 8

Generalizability

This chapter will examine the results derived in the previous sections of this thesis and attempt to draw general conclusions for object-oriented languages, abstracting from the specific language chosen. There exist numerous object-oriented programming and modelling languages, so that a compehensive review is beyond the scope of this chapter. For an overview of various languages and their respective features see e.g. (Capretz, 2003; Kirstensen and Osterbye, 1996; Wegner, 2003).

Any object-oriented language will exhibit a set of constructs to support the core principles, e.g. stimuli, call actions and reception in UML which support the communication by message-passing. However, each language will also have specific idiosyncratic constructs that may differ from language to language. Thus, any generalization of the results to other object-oriented languages cannot be focussed on specific constructs but must be oriented along the main principles of the object-oriented approach as discussed in Chap. 7.

The previous chapter has discussed a number of core principles of the object-oriented approach. These principles underly any object-oriented language. As our discussion in that section showed, the suggested rules and language changes do not violate these principles. Hence, object-oriented modelling languages that can express those principles, can be assigned similar ontological semantics. Insofar as the language does not depend on specific idiosyncratic constructs to support those principles, the ontological semantics can be transferred from UML to that language.

The formalization of the results in terms of OCL and the meta-model is

language-specific. Different languages may or may not provide a formalized meta-model in which to attempt formalization. Moreover, the meta-models of other languages may itself be expressed in other languages, not necessarily in UML, e.g. as generative grammars or as Entity Relationship Diagrams. However, the constraint language OCL is flexible enough to be usable in conjunction with such languages, the syntax and semantics of OCL and the usage of UML meta-model elements in OCL are independent of each other.

Tables 8.1 to 8.3 examine each of our proposed rules and corollaries as to whether they are applicable to object-oriented languages in general or whether they involve constructs that are specific to UML. This allows an assessment to what extent our rules are transferable immediately to other languages. Rules that make use of idiosyncratic language constructs cannot simply be transferred. Instead, the new language must be re-analyzed *in those parts*. Generally, it is not expected that any object-oriented language needs to be re-analyzed completely.

Table 8.1 shows that a large part of our interpretation is specific to languages that provide association classes as mutual properties are mapped to attributes of these. For object-oriented languages which do not provide this construct, another mapping must be found for mutual properties.

Furthermore, languages must support the modelling of attributes. As the representation of state information by attributes is not considered a central aspect of the object-oriented approach (e.g. Capretz (2003); Kirstensen and Osterbye (1996)) there exist languages which encapsulate state information and behaviour, without providing constructs that further reduce the state information to attributes. For such languages, another representation for ontological properties must be found, with different resultant rules.

As table 8.2 shows, some rules are applicable only to languages that have well developed state constructs, such as Harel state charts (Harel, 1988) rather than simple state-transition machines such as Moore or Mealy machines which do not provide constructs to express sub-states. As state charts or state-transition diagrams are not central to the object-oriented approach, there may exist object-oriented languages that do not provide any state constructs and are limited to descriptions of behaviour in terms of methods and operations. The mapping that we made in our analysis of UML leads to a set of rules and corollaries which mainly serve to ensure consistency between state charts and class diagrams. When a language does not provide state descriptions, such consistency constraints are not needed.

Table 8.1: Generalizable Rules (Static Structure)
(Gen.=Generalizability
X=unconditionally generalizable to other object-oriented languages;
(X)=generalizable conditional on existence of a particular language construct)

Rule	**Gen.**	**Comment**
Rule 1 (Substantial objects)	X	The object is the central construct to all OO languages.
Rule 2 (Properties as attributes)	(X)	The language must provide an attribute construct.
Cor 1 (Attributes are not things)	(X)	See above rule.
Rule 3 (Mutual properties as assocation classes)	(X)	Most OO languages possess association or relationship constructs, although not all languages permit attributes to be modelled for these.
Cor's 2, 3, 4, 5, 7, 8, 9 (Constraints on association classes)	(X)	See above rule.
Rules 4, 5 (Mutual properties and interactions)	(X)	While methods, operations and interactions are central concepts of the object paradigm, association classes are not.
Rule 6 (No composition)	(X)	Not all object-oriented languages make the distinction between aggregation and composition.
Rule 7 (Aggregates must have attributes)	X	
Rule 8 (Classes must have attributes)	X	
Rule 9 (Object ID's are not attributes)	X	While a distinct object identity has not been unanimously proposed as a fundamental principle, it is usually implicitly assumed.
		Continued on next page

From previous page		
Rule	**Gen.**	**Comment**
Rule 10 (Attribute values uniquely identify objects)	X	
Rule 11 (Attributes have values)	(X)	The language must provide an attribute construct.
Corollary 10 (Attribute multiplicities and the ordering of attribute values)	X	
Rule 12 (Additional behaviour or attributes lead to specialization)	X	Generalization and Inheritance are central principles of the object-oriented approach.
Cor's 11, 12 (Reclassification of objects)	X	See above rule.
Rules 13 (Aggregates must have two parts), 14 (Emergent properties lead to specialization)	X	
Rule 15 (Object creation as change of class)	X	Object construction is supported in all languages.
Cor 13 (Object destruction as change of class)	X	See above rule.
Rule 16 (No class scope attributes)	X	
Rules 18, 19 (Abstract classes and generalization)	X	Generalization and Inheritance are basic principles of OO languages.
Rule 20 (No ordinary associations)	(X)	Association class construct may not be provided by all OO languages.

Table 8.2: Generalizable Rules (Change)
(Gen.=Generalizability
X=unconditionally generalizable to other object-oriented languages;
(X)=generalizable conditional on existence of a particular language construct)

Rule	**Gen.**	**Comment**
Rule 21 (State defined by attribute values)	X	If a language provides both the attribute and state construct, this rule is applicable.
Cor 14 (Transitions change an attribute value)	X	See above rule.
Rule 22 (Sub-states require additional attributes)	(X)	Only applicable to OO languages that provide state chart constructs.
Cor's 15, 16 (Sub-states and concurrent sub-states)	(X)	See above rule.
Rule 24 (Action states are sub-states)		The distinction between states and action states is not made by all OO languages.
Cor 17, 18 (States, actions and activity diagrams)		See above rule.
Cor 19 (Partitions as parts of composites)		Partitions are not a fundamental OO construct.
Rule 25 (Quantitative behaviour as top-level state chart)	(X)	If a language provides state constructs, this rule is applicable.
Rule 26 (Top-level transitions correspond to operations)	(X)	If a language provides state constructs, this rule is applicable.
		Continued on next page

From previous page		
Rule	**Gen.**	**Comment**
Cor 20 (Objects have at least one operation)	X	
Cor 21 (Stable states in top-level state chart)	(X)	If a language distinguishes between stable and unstable states, this corollary is applicable.
Cor 22 (Top-level transitions associated with events)	(X)	If a language provides the event construct, this is applicable.
Rule 27 (Qualitative change requires operations)	X	
Rule 28 (Methods describe state charts)	(X)	If a language provides state constructs, this is applicable.
Cor 23 (Method state charts initial states)	(X)	See above rule.
Cor 24 (Method state charts and events)	(X)	If a language provides the event construct, this is applicable.
Cor's 25 (State charts' relationship to other change constructs), 27 (Methods and attribute value changes)	(X)	If a language provides the state construct, this is applicable.
Cor 28 (Reception and state charts)		A reception is a very specific construct that may not be provided by all OO modelling languages.
Rule 29 (State charts implement methods, not operations)	(X)	If a language provides the state construct, this is applicable.
		Continued on next page

From previous page		
Rule	**Gen.**	**Comment**
Rules 30 (Operation associated with reception), 31 (Reception, operation and unique event)		A reception is a very specific construct that may not be provided by all OO modelling languages.
Rule 32 (Qualitative change and orthogonal regions)		This is specific to Harel state charts.
Cor 29 (Objects are changeable)	(X)	If the language provides the state construct, this is applicable.

Table 8.3: Generalizable Rules (Interaction)
(Gen.=Generalizability
X=unconditionally generalizable to other object-oriented languages;
(X)=generalizable conditional on existence of a particular language construct)

Rule	**Gen.**	**Comment**
Rule 33 (Message passing requires association classes)	(X)	Assumes an association class construct.
Rule 34 (Objects send and receive messages)	X	
Cor 30 (Messages not between association classes)	(X)	Assumes an association class construct.
Rule 35 (Constraints within class)		Constraints may not be found in all OO languages.
Cor 31 (State transitions modify association class attributes)		Assumes both an association class construct and state transitions.
Cor 32 (Interaction is state transition in both objects)	(X)	Applicable if the language provides states and transitions.
Cor 33 (State transitions modify association class attributes)		See Cor 31 above.
Cor's 34, 35 (Signal and call events associated with top-level state transitions)		Different event types are specific to UML.
		Continued on next page

From previous page		
Rule	**Gen.**	**Comment**
Cor's 36, 37, 39 (Synchronous communication in state charts)		Not all OO modelling languages distinguish synchronous and asynchronous communication.

The analysis of rules and corollaries related to interaction (Table 8.3) shows that interaction in UML is expressed using a number of very specific constructs that are likely not generally available in all object-oriented languages. In addition, in our analysis we chose not to interpret messages as ontologically real. As a consequence, very few of the rules are generally applicable. Of those that are, the majority relate the message-passing interaction to state charts, as interaction is expressed as state changes in the BWW-ontology. This in turn requires not only that the language provides state and transition constructs but also association classes, as we have mapped mutual properties to attributes of association classes. Mutual properties in turn play a central role in interaction in the BWW-ontology.

To summarize, a substantial number of the guidelines is transferable to other object-oriented languages. Where they are not, the detailed discussion in Chapters 4 through 6 may be used as a guide for the re-analysis of particular aspects of the target language.

Chapter 9

Examples

This section provides two brief examples taken from textbooks on UML systems analysis. These examples are intended to demonstrate how models developed according to the proposed rules and with an ontological background differ from 'standard' UML analysis models. Such a demonstration serves to highlight two important results: (1) It shows a clear difference between two very different types of models. On the one hand, models generated with an implicit but not articulated background of IS development. These models are often termed analysis or conceptual models by their developers, but are produced before the background of IS design and implementation. On the other hand, models developed by specifically excluding any IS considerations and critically examining the real world. (2) It shows that the proposed semantics and rules are applicable in practice and that they lead to useful outcomes. Thus, practitioners are able to immediately put this research into practice.

Unfortunately, very few cases using UML are available in the literature which cover static and behavioural elements. The following examples focus therefore on class diagrams. These include operations assigned to class instances, so that a certain part of behaviour is modelled.

9.1 Example 1

This section will demonstrate the proposed semantics using an example taken from the systems analysis textbook by Hoffer *et al.* (2002) and the

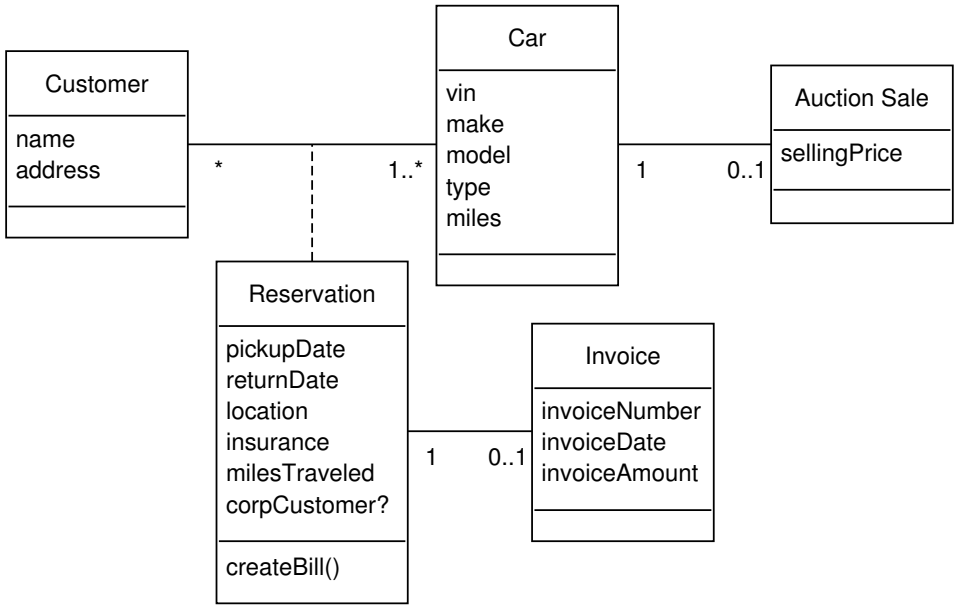


Figure 9.1: Car rental example class diagram, from (Miller, 2002)

instructor's manual (Miller, 2002). The next paragraph provides the description given in (Hoffer *et al.*, 2002), followed by an analysis of the given model vis-a-vis the proposed rules. We then re-analyze the case and derive a diagram which follows the proposed rules.

Description "An auto rental company wants to develop an automated system that would handle car reservations, customer billing, and car auctions. Usually a customer reserves a car, picks it up, and then returns it after a certain period of time. At the time of pick up, the customer has the option to buy or waive collision insurance on the car. When the car is returned, the customer receives a bill and pays the specified amount. In addition to renting out cars, every six months or so, the auto rental company auctions the cars that have accumulated over 20,000 miles" (Hoffer *et al.*, 2002). The class diagram, taken from (Miller, 2002), is shown in Fig. 9.1.

Analysis of Rules We begin by evaluating the class diagram according to our suggested rules and will then interpret the given real-world situation

in order to construct an ontologically meaningful model that conforms to the proposed rules.

The two classes 'Invoice' and 'Auction Sale' violate our rule 1 (substantial objects). Invoices and auction sales are events or interactions, not objects, even though there may exist documentation about invoices or auction sales. In our case, we are strictly concerned with the events or interactions. While the reservation class is a bundle of mutual properties between two things and thus correctly modelled according to rule 3 (mutual properties as association classes), it violates corollaries 5, 8 (associations classes have no operations and no associations) and the method 'createBill' should be modelled with the thing that actually does the invoicing, probably an employee or, more generally, the rental company (rule 4, changes of mutual properties). From the fact that the attribute 'corpCustomer?' is modelled, we assume that reservations for corporate customers are different from reservations for other, non-corporate, customers. Hence, according to rule 12 (additional behaviour leads to sub-classes) the corporate customers should be modelled explicitly. The associations between 'Auction Sale' and 'Car' and between 'Invoice' and 'Reservation' violate rule 20 (no ordinary associations) because they are not association classes.

A number of rules concern elements or relationships between elements that are not shown using the graphical UML notation and are only analyzable in terms of the meta-model elements. Adherence to these rules cannot be evaluated by examining the diagrams.

Ontological Analysis The original case description in (Hoffer *et al.*, 2002) does not provide much information about the real world situation. We therefore attempt to make reasonable assumptions as we proceed with the analysis.

The analysis and modelling of this example will be very much driven by examining the interactions. Rules 2, 3 and 5 (association classes represent mutual properties due to interaction) define the nature of mutual properties as arising out of interaction. Rule 1 (substantial objects) defines the possible interacting things.

In the real world, a customer interacts, by phone, fax, or computer, with some sales employee or sales clerk of the car rental company. We abstract from the medium of this interaction, which, according to rule 33 (message passing requires association class), should be included in a complete descrip-

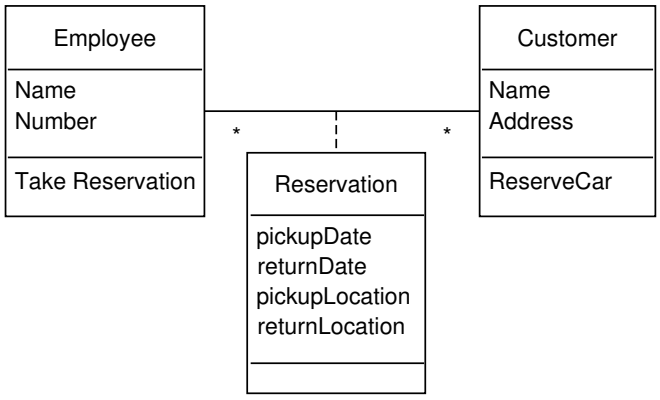


Figure 9.2: Car rental example: Class diagram, reserving a car

tion. The customer reserves not a specific car, but a type of car for a certain period. Thus, the event of reservation gives rise to some mutual properties between the customer and the car rental company, or more specifically, the sales employee of the car rental company. This is shown in the class diagram in Fig. 9.2. By rules 8 (classes possess an attribute) and 10 (attribute values identify objects) there must exist uniquely identifying attributes of the employee and the customer, modelled as name and address or name and employee number. As all action and interaction that is externally induced must be modelled as an operation (rules 26, 27, 30) we identify the actions that the customer and employee carry out as reserving a car and taking a reservation.

The customer and the employee share a number of mutual properties, i.e. the pickup date requested, the return date requested and the pick up location. These are mutual properties of the customer and the sales employee, modelled as attributes of an association class. As part of the activities of taking a reservation, it is now the sales employee's responsibility to schedule a car and the customer's responsibility to pick up the scheduled car.

Next, we assume that the employee of the rental company makes some arrangements to schedule a car of the proper type to be at the proper pickup location at the requested pickup time. An instance of class car that is scheduled acquires additional properties with respect to other cars, mutual properties with the employee that scheduled the car. By corollaries 11, 12 (acquisition of properties leads to re-classification) and rule 19 (special classes

define additional attributes) we must model scheduled cars as instances of a subclass of cars and model an operation to change the class of instances. In most cases the pickup date and location of the schedule match the requested reservation dates and locations by the customer.

The customer next interacts with the same (or some other) sales employee for the car pick up. This is an event which gives again rise to mutual properties, this time between the customer, the car that she picks up and an employee of the company. The car has been picked up on a certain date and the customer has requested of the sales employee certain insurance coverage for the car (Fig. 9.3). Cars that are picked up form a sub-class of the scheduled cars which can be returned and for which a mileage count per rental is important.

The next interaction occurs when the customer returns the car to the rental company. This interaction again gives rise to a number of mutual properties between the customer, the car and the employee, such as the date the car was returned, the number of miles travelled, etc. This is shown in Fig. 9.4. Note that a returned car is a special instance of cars and not of rented cars. This is because that car cannot be returned again, therefore it must not inherit that operation from the rented car. At this time, we have also renamed the renting customer to 'RentalCustomer' in anticipation of a purchasing customer later in the analysis.

The next interaction occurs between the customer and the employee of the rental car company. The employee provides an invoice to the customer. This interaction gives rise to mutual properties such as 'amount due' etc, which are captured by the association class 'Invoice' as shown in Fig. 9.5.

Cars are also sold at auction to customers. Every such sale gives rise to mutual properties between the car rental company (or an employee thereof), the car and the purchasing customer. This is shown in Fig. 9.6. We have modelled purchasing customers as a different subclass of customers as they have different (mutual) properties. We also include an operation with all customers by which change of classification (qualitative change) can occur. Similarly, an operation is included with all cars that expresses the qualitative change of a car when it is sold.

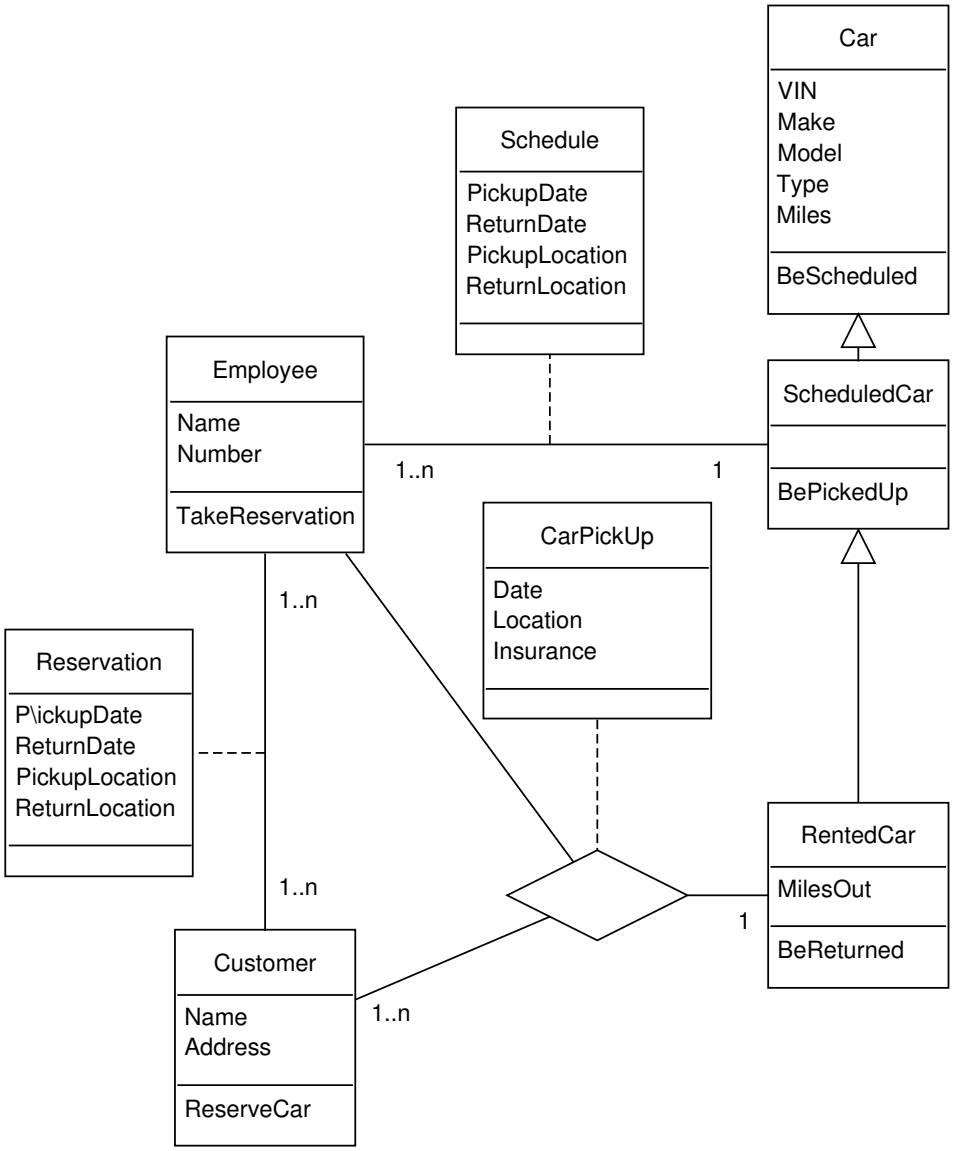


Figure 9.3: Car rental example: Class diagram, scheduling and pick-up

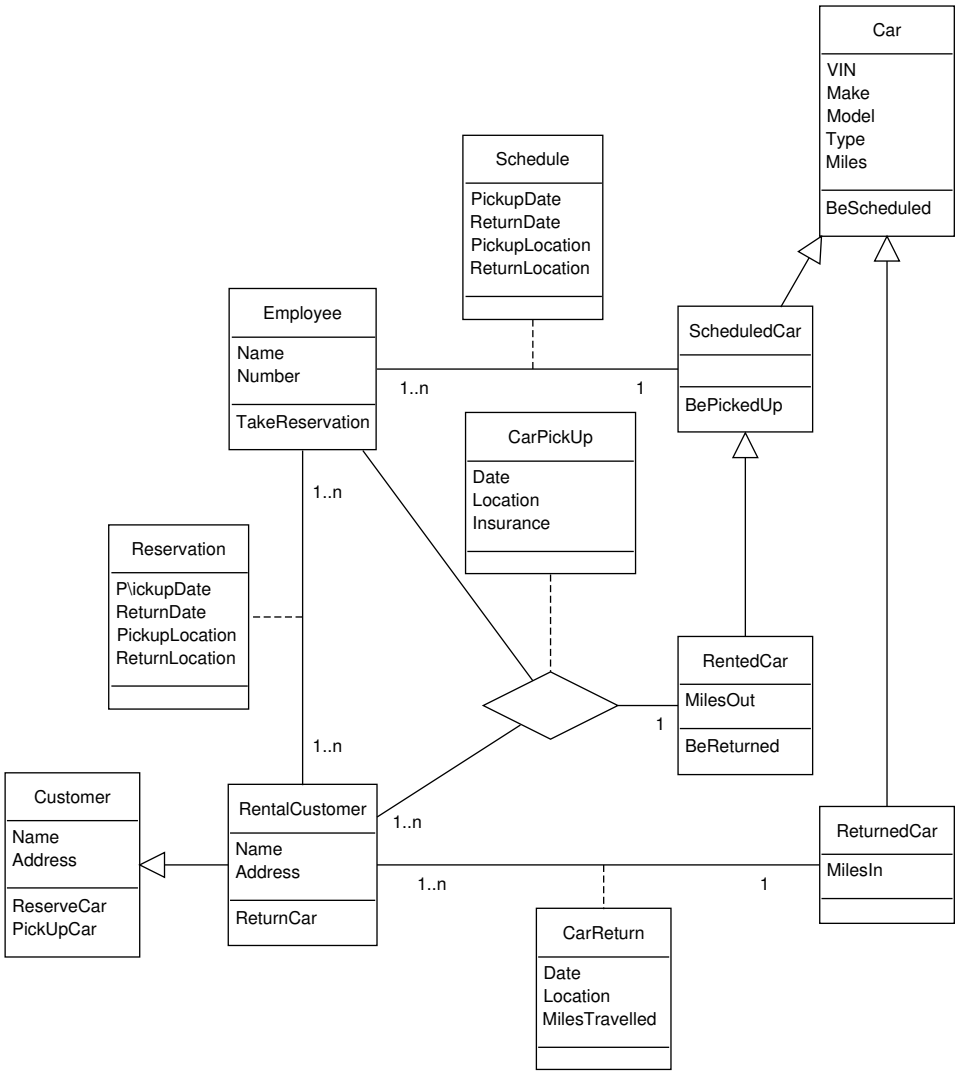


Figure 9.4: Car rental example: Class diagram, returning a car

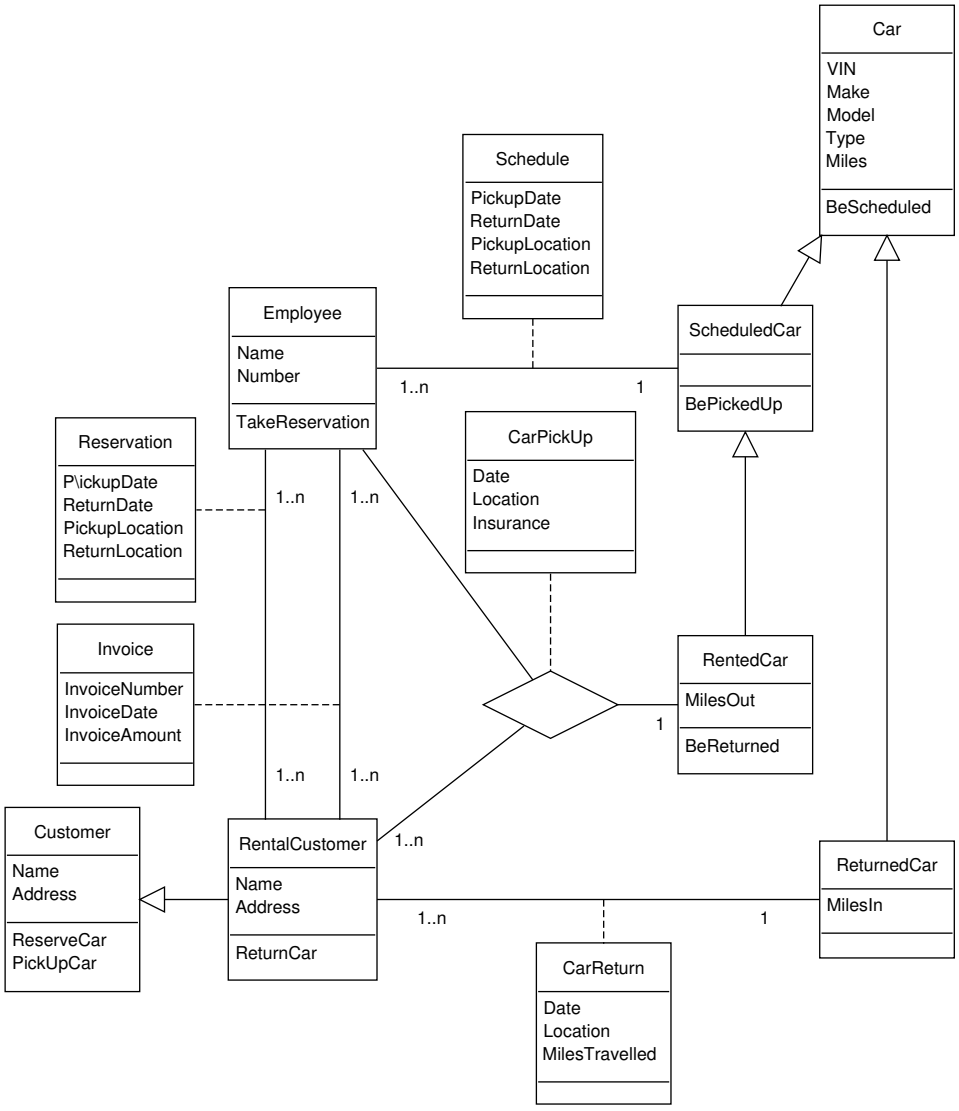


Figure 9.5: Car rental example: Class diagram, billing the customer

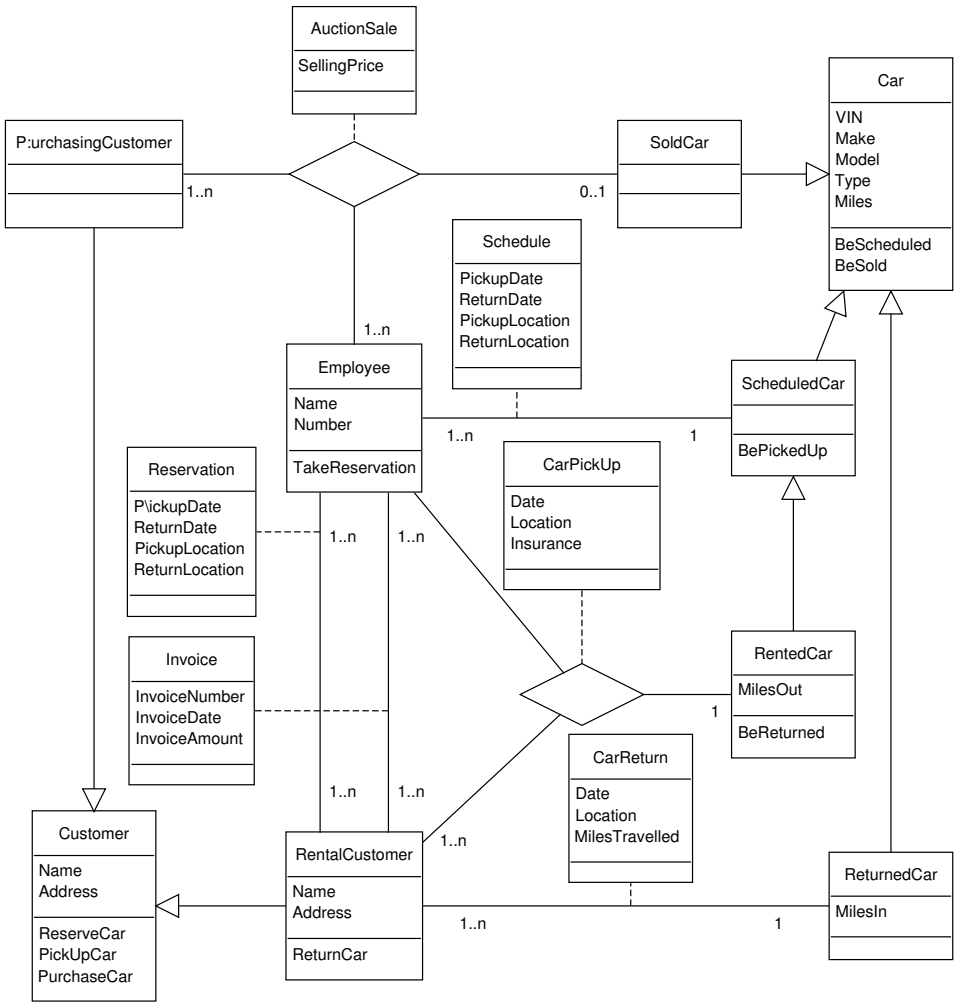


Figure 9.6: Car rental example: Class diagram, purchasing a car

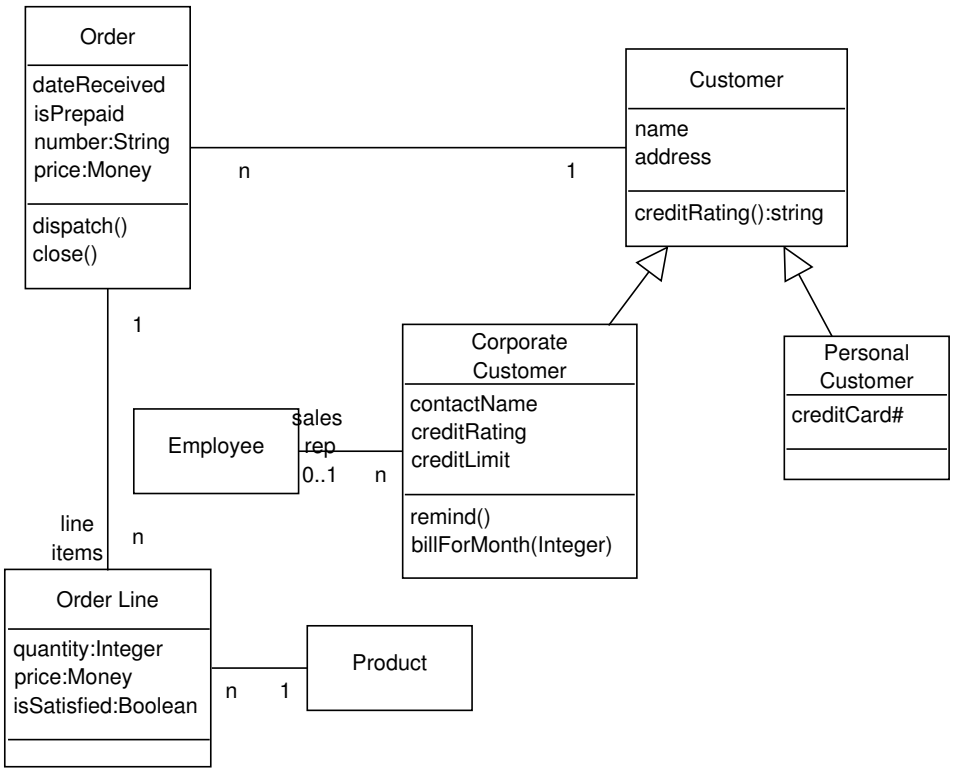


Figure 9.7: Example UML class diagram without ontological semantics (Fowler and Kendall, 2000)

9.2 Example 2

This section will demonstrate ontological analysis and the proposed rules using another small example, found in the popular UML introduction by Fowler and Kendall (2000) (Fig. 9.7). The book provides no case description. We begin again with an analysis of the rule conformance of the model.

Analysis of Rules There are a number of rules that are violated in various places of the model. Both the 'Order' and the 'Order Line' objects violate rule 1 (substantial objects) because they are not substantial objects but either an event (in the case of the order) or information, i.e. properties that arise because of interaction. Thus by 2 (properties as attributes) they

should be shown as attributes. Specifically, by rule 3 (mutual properties as association class attributes) they should be shown as attributes of an association class. Furthermore, by corollary 5 and rule 4 (changes in mutual properties by operations of regular classes) the operations currently assigned to the 'Operation' object must be re-assigned to the object that actually does the dispatching and closing of orders. Both the 'Employee' and the 'Product' classes violate rule 8 (classes possess attributes) as they possess no attributes. Moreover, they are not identifiable as they lack identifying attribute (rule 10). Note also that 'creditRating' is not something that the customer does, it does not even seem to be an action in the ontological sense that it changes the value of some property. Instead, the customer possesses a property 'CreditRating' which should be modelled as an attribute.

In order to address these issues, we need to re-analyze the actual real world situation that this model is intended to represent. Since only the diagram is given, there is a certain amount of subjective interpretation involved. The following sub-section presents one possible interpretation, in which a customer orders anonymous things. In another interpretation, the customer orders specific product instances. The latter interpretation is developed in appendix D.

Ontological Interpretation A customer orders a product from a company through an employee of the company. Note that in all cases the customer interacts with an employee, even though for personal customers this interaction is not with a dedicated sales representative. We assume that the model in Fig. 9.7 does not show the company as a class as employees are the relevant things. The customer ordering a number of products is an ontological event and must therefore not be modelled as an object of a class. Instead, the order event gives rise to a number of different mutual properties between the customer, the employee (or company) and the products ordered (rule 3, 5). As indicated in Fig. 9.7 these include 'dateReceived', 'isPrepaid' etc.

A naive first model like the one shown in Fig. 9.8 expresses the fact that a customer can order many products and a product (item) is ordered by exactly one customer. This is incorrect, because in this model there is no way to distinguish multiple orders by the same customer. In the real world, the customer orders a group of products per order, so the model should reflect this. Fig. 9.9 shows the correct model. With this model, each customer can order multiple groups of products, one for each 'Order'

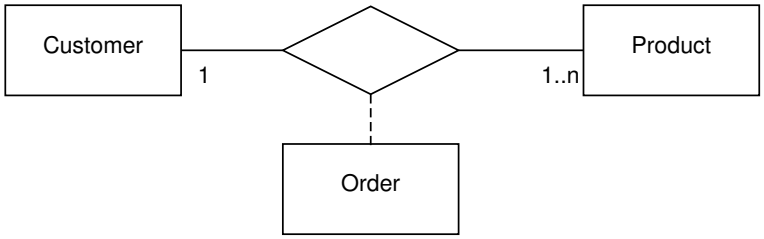


Figure 9.8: Order, incorrect model

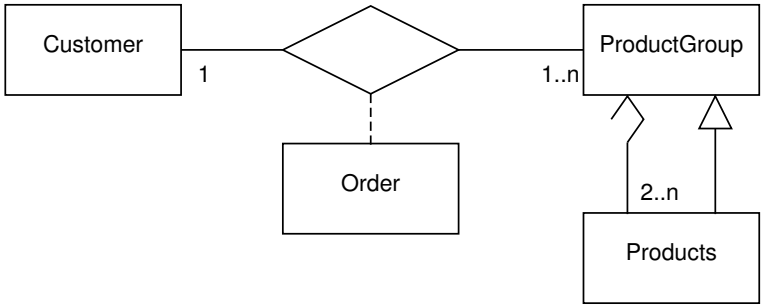


Figure 9.9: Order, correct model

interaction. Notice that because of rule 13 we must express the fact that a single product item can be ordered by modelling the fact that, while a product group cannot consist of less than two products, a product may be a product group.

We assume that the purpose of the orderline in the original model (Fig. 9.7) is to summarize the quantities ordered for different kinds of products. In other words, while an order refers to e.g. a total of 30 items of different types, there exists an orderline referring e.g. to 10 product items of type foo and another orderline referring to e.g. 20 product items of type bar. Product types are not ontologically substantial and cannot be modelled as classes. However, the 10 product items of a type foo form a composite thing which can be modelled as a product type group. We therefore refine the diagram of Fig. 9.9 into the model of Fig. 9.10.

Notice that just like an order pertains to group of products, each orderline refers to a group of products of a particular type, e.g. 20 items of type foo.

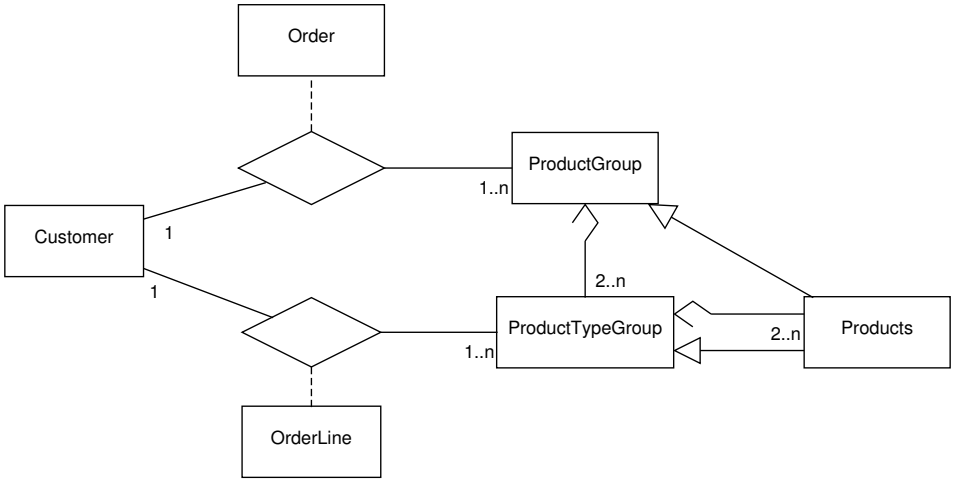


Figure 9.10: Product types

The original model in Fig. 9.7 showed the employee participating in an association with a corporate customer. Our rules suggest that every association must be an association class with at least one attribute. We suggest that sales representatives are specialized kinds of employees which possess mutual properties with corporate customers and therefore must be modelled as a sub-class of employees. Mutual properties could be e.g. contact name with the corporate customer, the account volume, account number etc. This is shown in Fig. 9.11. The diagram in Fig. 9.12 integrates this into a final class diagram, conforming to the proposed rules and developed using ontological analysis of the real world.

We are now in a position to fully integrate the model, together will all attributes and multiplicities. These satisfy the rules developed in Sec. 4. The final model with ontological semantics is shown in Fig. 9.12.

As indicated in Sec. 4.2.3 these indicate how many other things a thing shares mutual properties with. The customer may be involved with many different employees ordering many different product groups. However, the product group can only be ordered once (recall that we are talking about identifiable instances). Similarly, the customer can order multiple product type groups (the items ordered in a single orderline) and the employee can of course sell multiple of these groups. However, in this case too the product type group, the aggregate of individual products, can only be sold once.

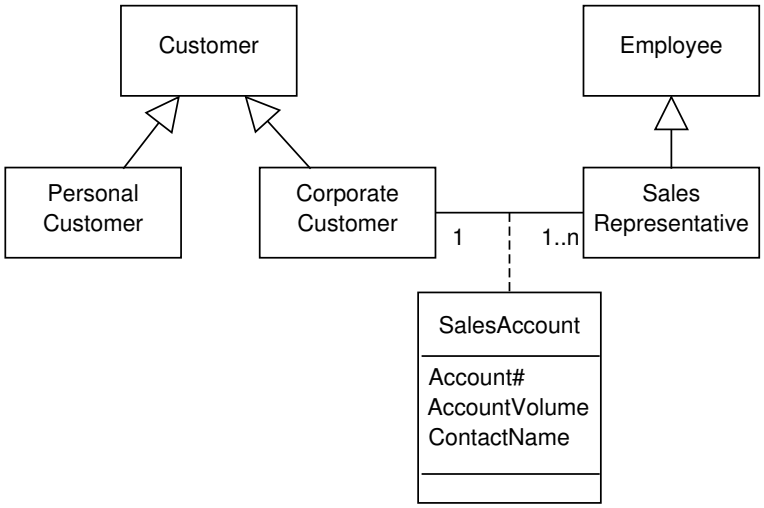


Figure 9.11: Customers and employees

Analogous to our reasoning in the first interpretation above, the methods have moved from the order and customer classes to the employee and sales representative classes, as it these that carry out the actions.

9.3 Discussion

This chapter has shown the applicability and the process of applying the proposed rules using two small examples. For both examples, a detailled analysis of the models showed that they violate a number of fundamental rules and, because of that, a number of corollaries following from those rules. Specifically, both examples showed that substantial things (e.g. customers) as well as events or interactions (e.g. reservations, orders) were modelled as ordinary classes or objects. This violates the proposed ontological mapping.

Both examples had to be re-interpreted as very little domain description was available. We have attempted to identify the real-world situation that they are based on and in the process have made a number of assumptions. Our proposed ontological semantics have been employed to reduce these ambiguities and to arrive at two ontologically meaningful models, corresponding to different interpretations of the original diagrams.

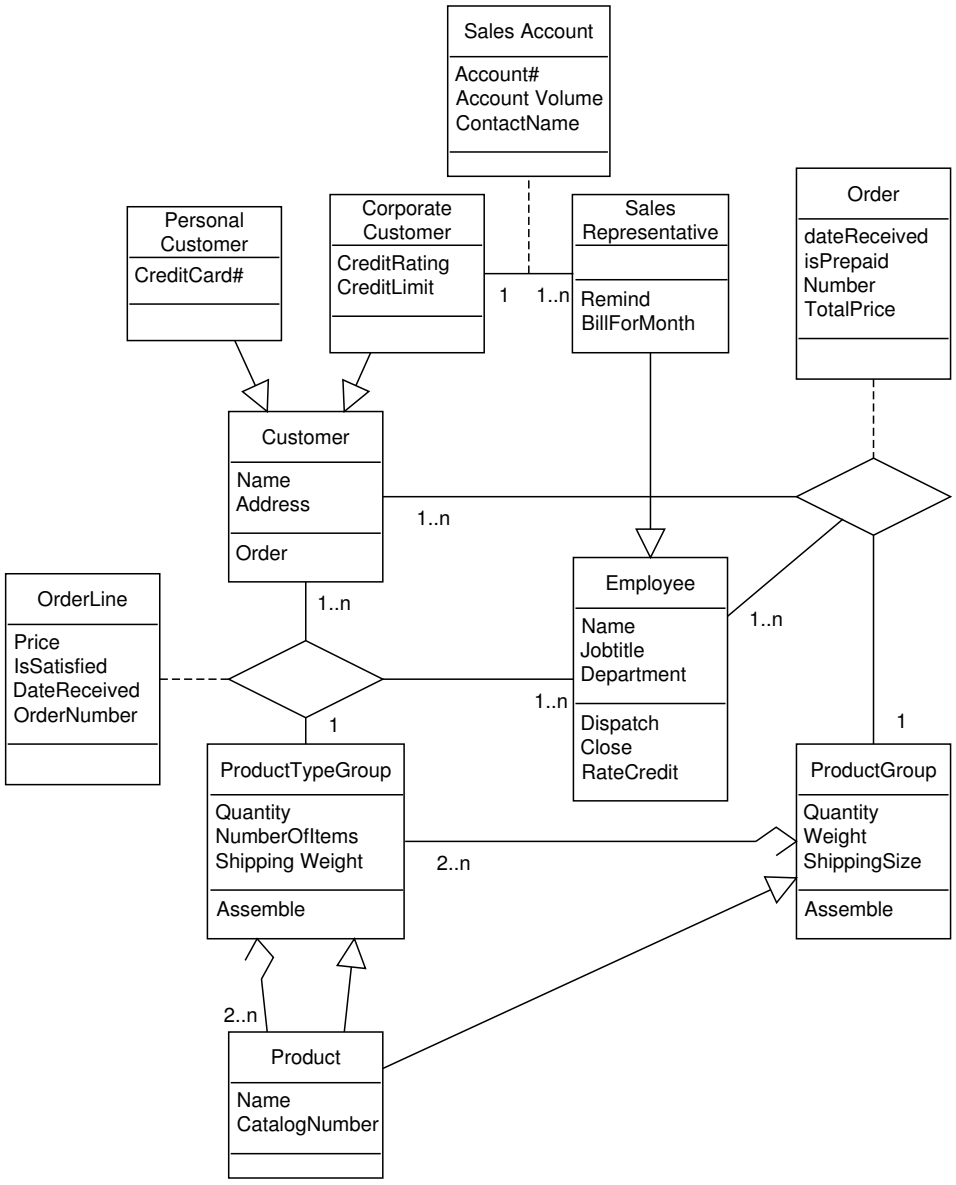


Figure 9.12: Final order processing class diagram

Not only are the initial models ambiguous (for the second example, an alternative interpretation is given in appendix D), but a comparison of the initial and re-interpreted models show that there is generally more detailled information present in the re-interpreted models. The process of re-interpretation was begun by an analysis of the real-world things and their interactions, as these interactions require and lead to mutual properties and thus association classes in the model. Following specific rules during the modelling process led to information to be included in order for the model to conform to the proposed semantics. For example, in the car rental example, this led to the explicit modelling of reserving, picking-up, returning and invoicing, while the original model assumed these steps only implicitly and cannot provide such a rich description of this. An example of additional detail in the order processing example, is the information about product and product groupings included in the model. This was implicitly understood in the initial model but not modelled with rich detail.

Levels of Abstraction Abstraction is the omission of details from a model or representation which are not relevant to the purpose of modelling. The intention is to ease the cognitive processing necessary for the interpreter of the model.

It is not the intention of the modelling rules or the two examples to increase the amount of information shown, even though in these cases that is the effect of the rules. The models developed in this chapter show more detail than the original models, by virtue of following the suggested rules and guidelines. It may be argued that not all of the information is relevant to the purpose of the model. However, no more specific purpose than a faithful representation of the real world has been suggested for these models. Thus, it is difficult to argue the relevance of individual model elements. Moreover, any abstraction should be made explicit, as a formal transformation of models.

For example, Fig. 9.1 and Fig. 9.6 differ in the way that properties relating to car pickup and car drop-off are modelled. In the original model, these are collapsed into a single class while the redeveloped model shows them as two distinct association classes. The distinction may be relevant to show that invoice numbers are only assigned once the car is dropped off. On the other hand, it is conceivable that invoice numbers are assigned at the time of reservation or at the time of pickup, so that this distinction becomes important. We therefore suggest that in a subsequent, explicit step, models

may be transformed by collapsing association classes between the same set of classes.

As another example, Fig. 9.7 and Fig. 9.12 differ in their treatment of products. The redeveloped model makes a clear distinction between product instances and groups of products. It shows clearly that an order line refers to a particular type of products, which the original model does not show. Again, an explicit model transformation should be used to collapse the redeveloped model of Fig. 9.12 to the one in Fig. 9.7 once it has been determined that, from a business or organizational perspective, it is not necessary to know that order lines refer to specific types of products.

With respect to the cognitive load on the model interpreter, only an empirical investigation can show which model is easier to understand or interpret. This will be examined further in chapter 11 below.

To summarize, the experience with these two small examples shows that (1) the rules are applicable in practice, (2) they lead to useful information being included in the model and (3) the rules allow the modeller to question and explicate assumptions about the real-world and the model. After these two brief examples, the next chapter provides, as a case-study, an analysis of a real project.

Chapter 10

Case Study

The aim of the case study is to provide corroboration of the feasibility of applying the derived rules and guidelines in practical situations, beyond the scope of small examples such as those in Chapter 9. This section shows that the proposed rules are usable for a medium size analysis project. Another empirical study, a controlled laboratory experiment, will show the specific beneficial effect of the rules, an increase in analysts understanding of the domain (Chapter 11).

10.1 Organizational Setting

The organization under study is a large North American university. The university is concerned about the image it presents to prospective students and has undertaken a business process re-engineering (BPR) effort to investigate streamlining the processes involving current and prospective students. This BPR led to a number of requirements for individual processes from which IS requirements were derived and appropriate projects initiated.

One of the identified projects is the provision of an opportunity for prospective students to assess their likelihood of admittance without having to initiate the official and lengthy admissions process. This is to be done by allowing prospective students to enter their educational history into an IS. The IS will then apply the university admission rules to this data and report the result to the prospective applicant, at which time the applicant may be allowed for self-admittance to the chosen program.

This project was chosen as the basis for the case study for two reasons: (1) It was a project that was nearing completion. The main stakeholders and informants of the analysis were still available for interviews. Their involvement with the project and the analysis was recent so that rich data could still be gathered. (2) The project team had used UML for system analysis. It was hoped that additional insight could be gained by comparing the project team's UML models with those developed as part of this case study.

The primary contact person is the co-project manager and lead analyst (LF). LF explained that, for his project staff, it is the first project to use UML. The use of UML in the project is restricted to the analysis and modelling of the real world domain using a UML class model. According to LF, this model is intended primarily for clarification and understanding of the real-world domain and contains the core classes to represent the organization. Subsequent design and implementation will be done with reference to the analysis model but without re-using it or using it directly for automatic software design or code generation. When this case study was begun, the project was almost at the beginning of the coding stage.

10.2 Procedure

As part of this case study, an independent analysis was undertaken in order to model the domain of student admissions and assessment. Interviews were conducted with relevant stakeholders and informants (see Appendix E) and UML models (class and sequence diagrams) were constructed. All interviews were conducted on-site and were using open-ended questions. The independent analysis serves two purposes: (1) It shows the applicability of the proposed modelling rules and (2) produces an ontologically meaningful alternative model which can be compared to the original one.

The proposed rules were used as guidelines to the analysis and the interviews. For example, rule 1 suggests that the analyst determine all physical things involved in the domain, while rule 2 requires the identification of properties of these things. For rule 5, the modeller must identify interactions in the real world in order to properly model association classes and their attributes. Rule 10 suggests that behaviour must change the thing's states (corollary 14). Thus, the analyst must identify sufficient attributes which change, in order to express operations or state transitions (rule 26). Once

interaction is identified, the modeller must ensure that association classes are defined according to rule 33 and all objects are involved in interaction (rule 34).

While the rules mentioned in the above paragraph are only a sample of all proposed rules, they are fundamental rules that can guide the modeller to seek required information about the real world system under consideration. However, this guideline must not be mistaken for a methodology or a process of object-oriented analysis of real world domains. Research by Wand and Woo (1999) provides such a process which also focusses on interaction. While not strictly followed in this research, it was taken as another guide to discover the elements of the organizational domain. Wand and Woo (1999) suggest that analysis begin with objects external to the domain and their interactions or requests. Such requests in turn lead to other interaction among objects within the analysis scope. The sequence of interactions should be traced to discover all relevant objects, behaviour and attributes.

The information offered by different project participants during open ended interviews is available in Appendix E. It was used for analysing and modelling the real-world domain. This is described in Appendix F. The following section discusses the experiences gained from the case study and compares the initial model by the project team with the models that adhere to the proposed rules, as developed in Appendix F.

10.3 Discussion

This section discusses and summarizes the findings of the case study. It begins by examining the process and the resulting models and then summarizes the discussion of the resulting models with two of the project team members. Figure 10.1 shows the class diagram developed by the project team. It is shown for comparison of the models and their features. The independently developed models are shown as part of the analysis in Appendix F.

Model Features The process of analyzing the domain and modelling it according to the suggested ontological semantics for UML shows that the models produced are valid UML models. All produced diagrams are valid UML models, and adhere to all regular syntactic rules. The models are not trivial and do not appear overly complex either. Hence, use of the rules is at least possible in a practical setting.

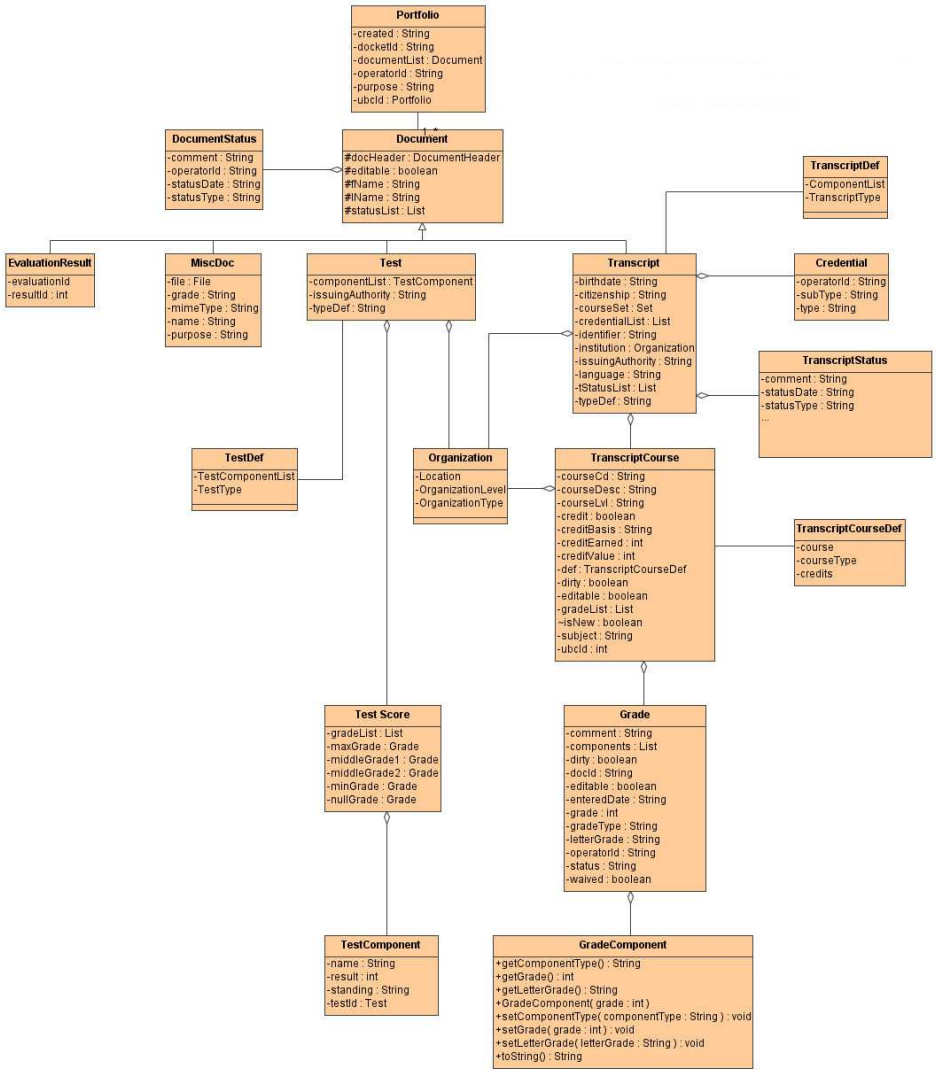


Figure 10.1: Class diagram developed by project team

The process of analysis and modelling has shown that at various stages in the process the rules and proposed semantics lead the modeller to include information in the model that is not necessarily relevant to building an information system, e.g. the interactions in high schools that give rise to the grades (e.g. Fig. F.1), but that are important to properly understanding the domain. It may turn out at a later stage, that information about the high school teacher becomes data of interest to be included in the information system. Another example of additional information in the model is the differentiation between the actual in-class standing, as defined by the interaction between teachers and high school students, and submitted grades, as reported by the school or the applicant to the school. This separation allows explicitly for the notion of incomplete or erroneous submission or reporting of applicant standing to the university.

As argued in Chap. 9, the inclusion of this additional detail is a by-product, not the intention of the modelling rules. Again, an explicit model transformation should be used to abstract from irrelevant model elements, rather than implicit considerations during model construction. For example, the fact that the re-developed model includes information about high school teachers and high school counsellors submitting grades to the university, may be important to not only maintain contact information about them, but also allows the university to target teachers and counsellors based on the quality of past students when promoting the university. If this is not currently relevant, there should be an explicit and justified decision to exclude this information during a subsequent model transformation step.

The inclusion of information about the admissions officer is another example. It allows the university to capture information for example about the workload of admission officers or to track information about the admission officers sending and receiving correspondence and making decisions about applicants. This type of functionality has been requested by some of the interviewed staff, but is not part of the current information system or the plans for the next version of the system. This information has been omitted from the original model without an explicit and well documented decision.

Another difference is the separation of association classes, e.g. between the applicant and the university. There exist the "Application" and the "Acknowledgement" assocation class (Fig. F.3). These mirror the interactions modelled in the sequence diagram. This specific way of modelling also contains additional information: It becomes clear which attributes belong to an Application and which attributes belong to an acknowledgement. This may

be important when only part of the model is relevant for IS development or different parts of the domain are to represented by different information systems.

With respect to action and interaction, the general absense of detailled state charts indicates mostly qualitative interactions. Given also the experience from the two short examples in the previous chapter (Chap 9), it can be surmised that non-technical, human organizational systems, generally involve more qualitative than quantitative change. The model shows that a number of configurations of the applicant are modelled as sub-classes that allow for qualitative change (e.g. Fig. F.15). This way of modelling introduces a large amount of additional information into class diagrams. One can almost follow the "life" of a high school student along the specialization associations, from student, to applicant, to accepted applicant, etc. This information is only implicitly contained in the original model. There, it is encoded in attribute values that are not explicitly shown in the model, and thus not obvious to the interpreter.

When quantitative change is modelled, e.g. when the state of an applicant changes due to changes in the admission criteria throughout the admissions process (Figs. F.12, F.13), application of the rules led to a fundamental insight into the domain. Instead of assigning states such as "on hold", "pending", "admitted", etc. to the student, these rules lead us to assign such states to the university. Furthermore, for each student, the university possesses an independent sub-state machine. The student's state cannot change without interaction, and the changes to admisssion criteria do not affect the student, only the student's admissability with respect to the university. Hence, these states must be states of the university. This confirms the notion that in the real-world, it is the university which manages the state information of the students and which can change the state information. Thus, modelling the states in this way leads to a much more realistic model. However, it may not be intuitively obvious to IS designers or programmers.

There are two interesting things to note when examining operations modelled for various classes. The initial model by the project team does not provide any operations. Attributes are either publicly visible and directly changeable or assumed to be provided with accessor methods (Get/Set). Either approach hides the semantics of the changes inside any method implementations. However, these are not part of the conceptual model. Hence, this information is lost to the conceptual modelling and understanding pro-

cess, and only implicitly assumed by the IS designer. The independently developed model shows operations for all object classes. It therefore becomes clear what an object of a certain class is capable of. Moreover, as these operations are related to external stimuli, modelled as messages in the sequence diagrams, the ordering of the changes expressed by methods can be deduced. This is information lacking in the initial model. Making this information explicit in the conceptual model furthers not only understanding, but also allows critique and assessment of the correctness of the information.

Model Assessment The discussion with two project team members, the project lead (LF) and the lead developer (CH) of the project, confirms these observations. They suggested that the alternative, independently developed model may indeed contain more information about the real world domain under study. Both team members commented on the fact that the original model contains many hidden and implicit assumptions. This is exemplified by the following quotes:

> *"We relied a lot on assumptions that were never written down in the model ... yours is more comprehensive."* (CH)

> *"ours have all sorts of stuff around that is assumed but not modelled"* (LF)

This was presumably unproblematic in the project as it was a relatively close group of analysts and developers, that were co-located and in constant communication. As the project lead had been involved in the preceeding BPR project that led to the project and system requirements, he was intimately familiar with the business and organizational domain. Furthermore, the project team did not undergo any changes throughout the project.

If the project team had undergone any changes during the development process, both interviewees felt that the alternative models would provide better understanding to a new team member or somebody unfamiliar with the domain.

> *"yours that you have are arguably better as an initial introduction"* (LF)

"It shows a better big picture view of how it fits together." (CH)

This may in part be due to the additional information in the model. This is contextual information, i.e. information about the aspects of the domain outside that which is to represented by the information system. As such, this information would be useful in allowing new members to discover why the requirements for the information system are defined one way or another, even though they do not themselves form requirements.

The independently developed model was generally felt to be complete and comprehensive, not only with respect to additional information, but also with real-world aspects that were modelled in the initial model by the project team:

> *"Certainly more comprehenssie there [in the areas outside the IS scope], but even in the smaller, there's a somewhat simpler, more elegant view in a few cases"* (CH)

The project lead commented on possible abstractions by sub-classification and specialization that he identified as missing in the independently developed model:

> *"there is a more generic, archetypical diagram..."* (LF)

This seems to suggest that the analysis either did not identify potential sub-classes and specializations, or that such abstractions are only possible due to the lesser amount of information in the initial model of the project team. This was the only aspect of the model that was deemed incomplete by the project staff.

In this context it is interesting that specialization and sub-classification are interpreted as advanced features and not present in the model developed by the project team:

> *"Inheritance also gets shied away from .. This may be because it is initially confusing to developers, they may not understand it properly"* (CH)

Similarly, more complex associations (of ternary or higher arity) and association classes are not always to familiar to developers.

"The large diamond sort of throws me off. Don't know directly what we're implementing there." (CH)

This may either be due to the inherent complexity of UML in terms of the large number of constructs of the language, or, more likely, due to the traditional database background of the developers:

"the fact that people were familiar with the database structure, this influenced things." (CH)

"There's a real danger of seeing object modelling as data modelling or relational modelling, as rows and foreign keys." (LF)

The second important purpose for a conceptual model besides the representation of a real world domain is the starting point for IS design and software development. The alternative model was not seen as defective in this way, it could serve this purpose:

"I don't see any reason why you couldn't just take these [the models] and run with them." (CH)

However, it was commented that it shows complexity and additional objects beyond what is represented in the original model which would make it less suitable for software design. Clearly the original model was less complex and modelled certain entities as attributes rather than object classes which is easier for software developers to translate to programming code:

"It [the new model] shows a lot of stuff that is not implementation related" (CH)

During the discussion with the project lead it also became clear that the use of UML without any guidelines is a big challenge in the project. Some analysts are aware of the subtle differences between possible interpretation and models and the difficulties associated with them, exemplified by the following quotations:

"It's normally difficult to model a course object, because it is a relationship ... What do you mean by a course? The curriculum, the interaction, the grade?" (LF)

"... presumably needs more thought to distinguish different kinds of objects." (LF)

"Modelling of non-substantial objects happens quite easily and naturally. You don't have to worry about empirical reality" (LF)

These quotes show the relative difficulty of discerning various features of the real-world and translating these features into UML models. In contrast, modelling events and interactions also as UML objects appears to be the easy way out of this difficulty. The project lead and lead developer agree that modelling rules are necessary and helpful, both for guiding the modelling process and for ensuring model quality and consistency:

"Such rules would have helped in our groups. The rules would tell whether a model is good and can help answer some questions. They seemed like a lot of valid questions to ask." (CH)

"Rules can force the modellers to think deeper about what they're modelling" (LF)

Another interesting finding is that the extensive use of sequence diagrams and modelling of interactions, which is a central aspect of the proposed ontological semantics and rules, is viewed as something that should be done more often in projects but is not.

"They don't get done as official project documents. When developers meet, two thirds of them will use inofficial sequence diagrams ... That makes things much clearer." (CH)

This suggests that the interaction centered perspective enforced by the proposed guidelines and re-inforced by the chosen approach to modelling (Wand and Woo, 1999), can make a positive contribution to understanding. Interaction diagrams appear as inofficial tools, yet interactions are not officially documented. Explicit modelling of interactions, forced by the use of the proposed rules, can help with understanding of the model and the domain. This supports the modelling approach used and also confirms the importance of interactions in analyzing a real-world domain.

Model Quality Chapter 1 positioned this research in the wider context of model quality. While this wider issue of model quality is outside the scope of this thesis, a thorough discussion must be left for further research. However, this section will highlight a few key issues framed by the three prominent frameworks of model quality which have been suggested by (Krogstie *et al.*, 1995; Lindland *et al.*, 1994), (Moody and Shanks, 1994, 1998; Moody, 1998) and by (Becker and Schütte, 1995, 1996; Rosemann and Schütte, 1997; Schütte, 1998; Schütte and Rotthowe, 1998; Rosemann, 1995; Schütte, 1999). One of these is the adequacy of the language for the modelling task citep-Schuette:99 which is the focus of this study.

During the course of the case study, three specific model quality dimensions could be assessed. The semantic quality (Krogstie *et al.*, 1995) or correctness (Moody and Shanks, 1994; Becker and Schütte, 1996) of the model with respect to the domain, appears to have increased. Based on the interview results, the project participants felt that the models contained more detail about the domain and thus provide a better picture of it.

The model retains its implementability (Moody and Shanks, 1994) in software. While the resulting software may be different in terms of time or space efficiency and other criteria, the lead developer felt that the model could be used for software design and software implementation. Specific and well- defined transformation step may be helpful at this stage, for example to abstract from detail not needed for software implementation, to adapt to a particular software or database framework, or to enhance the efficiency of the resultant software system.

Integration citepMoodyShanks:94 or systematic construction (Becker and Schütte, 1996) is increased because the proposed rules are often inter-diagram in their nature. Thus, the resulting models are increasingly well-integrated and coherent.

Possible effects on the cognitive quality dimensions of pragmatic quality (Krogstie *et al.*, 1995), understandability (Moody and Shanks, 1994) or clarity (Becker and Schütte, 1996) can be drawn from the interviews with the project lead and lead developer. It was felt that the models are relatively difficult to interpret, compared to the original model by the project team, and required some clarification in preparation for discussion. This may be due to their size and complexity, which is a function of their correctness viz-a-viz the domain.

Finally the benefits of the model must be discussed with respect to pos-

sible costs. These include but are not necessarily limited to learning the modelling rules, applying them in the construction of the model and interpreting the resulting models. The large number of rules and corollaries increased the amount of time necessary for the modelling process. The added size and complexity of the models increased the amount of time and effort needed to fully interpret and comprehend them. More quantitative measures of costs and benefits are discussed in Chapter 11 which describes an experimental study.

Conclusions In conclusion, the case study shows clearly that the rules and semantics are applicable to a practical real world situation. This is shown by the results of the analysis (Appendix F) as well as the discussion with the project team members in this section.

The resulting model was deemed more comprehensive, more understandable and more exact with respect to it's role as representation of the real-world domain. These qualities may make it better suitable also for new team members to comprehend. This is important as IS development teams are generally comprised of members from diverse backgrounds and organizational units with potentially very little exposure to the particular problem domain.

The resulting model is also suitable for software development, although it is more comprehensive than necessary for this purpose, by reducing the information in the conceptual model. It may therefore be worthwhile to develop intermediate transformations which would facilitate this purpose. Any such transformations should be well defined and explicated.

While the resulting model appeared more comprehensive than the original one developed by the project team, quantifiable estimates of effort for the original model could not be obtained, so that a comparison in terms of modelling effort or a cost/benefit analysis is not possible.

In summary, the case study supports the use of the proposed rules and guidelines and confirms their usefulness in understanding the domain and communicating that understanding through conceptual models.

Chapter 11

Experimental Corroboration

Chapters 4, 5 and 6 outlined the development of real-world semantics and rules for using UML for conceptual modelling. Chapters 9 and 10 showed that the rules can applied and the resultant models are at least sensible. We showed how the rules allow the modeller to focus on the real-world aspects and guide the ontological analysis process of the real-world domain.

However, it must not only be shown that the rules are actually applicable and feasible, but also that the resultant models are better. This chapter outline our understanding of 'better' and develops and experimental tests of the proposed rules show the improvements in quality of conforming models over non-conforming ones.

One of the main purposes of a conceptual model is to serve as a *communication* tool among the participants of the ISAD process (Kung and Solvberg, 1986) to help arrive at a *common* understanding or agreement on what constitutes the problem domain (Schütte and Rotthowe, 1998). Hence communication is successful if the receiver of the communicated information gains the same domain understanding that the sender of the communicated information possesses.

Every communication occurs by some medium. In our case the conceptual model serves as that medium. The sender encodes the message on the medium, in this case by formulating or building a model using a certain language. The receiver must then decode and interpret the model.

Norman (1986) proposed a theory of action for the user interaction with a technological device such as a computer. This theory suggests that for

the user to make sense of readouts of the device, the representation must be in a form that corresponds to the user's mental model. This allows the user to bridge the *gulf of evaluation* more easily than in situations where the representation does not conform to the user's mental model. On the other hand, for the user to act on the device, the interface should again be in a form that the user is familiar with. This allows bridging of the *gulf of execution*, i.e. the user plans to execute some action on or using the device, hence must translate from his own conceptual schema to the representation or interface the device presents. Gemino (1999); Gemino and Wand (2001) suggest that Norman's theory of action (Norman, 1986) is also applicable in the context of conceptual modelling, by suggesting that the model in this case assumes the position of the representation of the device. While this is a broad interpretation of Norman's theory, it serves to illustrate the two aspects involved in modelling.[1]

A complete test of our theory involves examining the entire ISAD process from model construction to model interpretation. However, since examining both of these aspects together introduces confounds into the investigation, these tasks must be examined separately (Gemino and Wand, 2001). This study focuses on model interpretation, leaving model construction for future research.

11.1 Theoretical Model and Hypotheses

We make the assumption that ontology is not only what exists in the world, but moreover, ontology is expressed by what humans agree exists in the world. We claim that human cognition reflects ontology, we do not make the inverse claim that human cognitive constructs are our ontology. I.e. we do *not* take an idealist position. The true idealist will find the following paragraph of justification unnecessary.

As realists, we believe that this is a sensible position to take. Being situated in the world, human beings are continuous problem solvers (Newell and Simon, 1972) (either everyday tasks or solving scientific problems) and hence require mental models and theories of the world. Thus, over time the mental model will come to correspond with the real world and ontology becomes what we believe to exist in the world. There is no need to suggest

[1] One can of course think of the modelling language of the device. This may especially be true in the case of CASE tools.

innate ontologies, the notion of feedback suffices for this argument.

A well accepted model of the human memory is that of a semantic network(Collins and Quillian, 1969, 1970; Collins and Loftus, 1975) with concepts as nodes being connected by associations as edges. Recall from such a network is by *spreading activation* (Collins and Loftus, 1975; Anderson and Pirolli, 1984; Anderson, 1995). A semantic network supports reasoning because the associations among concepts allow inferences to be drawn[2]. A specific ontology such as Bunge's then serves as part of the semantic network. It defines concepts such as thing and property which are associated with one another.

Learning and understanding is the integration of new concepts or new associations into such a network[3] (Anderson, 1995). The interpreter of a model must incorporate the mental concepts representing the model elements into her mental schema. If the model follows ontological semantics it will exhibit certain features such as things and properties that correspond to already existing ones in the interpreter's mental network. Hence, this will facilitate integration. On the other hand, if the model does not follow such semantics it will exhibit model elements that contradict or are incompatible with the existing mental network, thus leading to improper or no integration at all. When the model is well integrated into the existing semantic network, it is possible for the interpreter to use pre-existing associations between concepts for reasoning beyond the information contained in the model. If the model is not integrated into the semantic network, such reasoning will be more difficult.

Solving problems requires drawing inferences based on existing knowledge, i.e. reasoning (Newell and Simon, 1972; Anderson, 1995). Hence, we propose that problem solving can be used as a way to measure the amount of reasoning that a given mental network supports. It is thus a measure of domain understanding (Gemino and Wand, 2001).

To summarize, we believe that interpreting a model based on the sug-

[2]This model also accounts for a number of observed cognitive phenomena (Hirsh-Pasek *et al.*, 1993) and is compatible with new theories of meaning. These are often called "theory" theories as they posit that the meaning of any concept is determined by a theory which incorporates this concept (Gopnik, 2001). Such a theory is easily expressed as a semantic network. Moreover, this model is compatible with ideas of coherence theories of meanings put forward in the philosophy of language (Quine, 1953). These suggest that a words meaning is determined by the way it fits into a coherently structured system of relationships to other words.

[3]Or the strengthening or weakening of associations.

gested semantics leads to better domain understanding as measured by problem solving ability:

Hypothesis 1 *Interpreting a diagram conforming to the suggested real-world semantics and rules will lead to better performance on* problem solving tasks *than interpretation of non-conforming diagrams.*

11.2 Prior Research

In order to build on the strenghts of past research and identify previously used instruments, this section reviews empirical research related to assessment of the relative performance of modelling languages or modelling techniques. Research into both model construction and model interpretation is examined.

A recent literature survey (Johnson, 2002) reviews a number of empirical studies related to object-oriented systems analysis, lamenting a general lack of theoretical foundations and poor experimental designs combined with small sample sizes. This is exemplified by a number of prior studies. Wang (1996) uses 44 student subjects and a representation task to assess advantages of object-oriented versus data flow (DFD) methodologies. No theoretical model is provided to indicate why such performance differences should arise. Subjects each had three modelling sessions and the models were assessed in terms of syntactic correctness and semantic accuracy. Results show that the method had no impact on syntactic correctness but semantic accuracy was greater for OO than DFD models.

Earlier work by Yadav *et al.* (1988) compares the construction of DFD and IDEF0 models. DFD and IDEF0 are very similarly in their modelling capability and method. Yadav *et al.* (1988) compare the product as well as the process of modelling, but do not suggest an explanation why the techniques should differ. There were significant differences in the modelling process (learnability, ease of modelling) but no differences in model correctness, completeness, consistency and appropriateness.

Batra *et al.* (1990) examined model correctness versus the original textual representation. The study attempted to rule out possible ambiguities by pilot testing the experiment until no ambiguities remained in the problem description of the possible model. It was then possible to assess model correctness as representational accuracy. The study by Vessey and Conger

(1994) focuses not on the model or process but instead on the learnability of techniques. They employ process tracing techniques to compare JSD, structured analysis and object techniques.

There are two things to note about these early studies. The representational accuracy is a prominent dependent variable, suggesting that this research recognizes the importance for a model to be a good representation. Furthermore, all of the studies recognize that representational accuracy cannot be objectively assessed, and so expert panels and expert judges are employed to rate the models.

More recently, Bodart *et al.* (2001) argue for the use of sub-typing and against the use of optional properties. However, their chain of argument depends on the selected language feature (in this case optional properties) and is not immediately generalizable to the overall language in terms of its ontological properties. The important relationship to the present work is their use of conceptual networks and spreading activation to motivate differences in language performance. This lends support to our theoretical basis discussed above.

A study by Kim *et al.* (2000) suggests that the integration of multiple diagrams can be facilitated both on the perceptual level and the conceptual level through appropriate diagrams. The experimental study confirms the hypothesis that visual cues and contextual information does indeed improve integration of multiple diagrams. This study is particularly relevant as it provides support for the idea that contextual information beyond the diagrams themselves can help diagram integration. The theoretical results derived in earlier chapters facilitate the integration of different diagrams in a conceptual whole, by relating them to a single ontological model. One would expect that diagrams conforming to ontological rules can provide better integration of the structures not only within a single diagram but also across multiple diagrams.

Cognitive Fit and the Human Problem Solver One widely used theoretical foundation for comparing different languages is the theory of cognitive fit (e.g Vessey, 1991; Agarwal *et al.*, 1999; Sinha and Vessey, 1992). This model was developed in IS language research beginning with (Vessey, 1991) and is based primarily on the arguments of (Newell and Simon, 1972). According to it, better task performance results when the representation of the problem solving task matches the representation of the problem. The problem solver can use processes that emphasize the same type of informa-

tion that is predominant in both the problem representation and the task and can use these processes to operate on the *internal* (mental) representations of the task. This has been interpreted as the assumption that a fit between the modelling language and the problem description can facilitate modelling performance.

Accordingly, the work by Sinha and Vessey (1992) tests problem solving behaviour given a task and a representation. Comprehension or understanding are not part of the model. Agarwal *et al.* (1996) examine model construction within the context of Vessey's cognitive fit model and later also examine model comprehension (Agarwal *et al.*, 1999). Both of these studies compare procedural with object-oriented techniques. In (Agarwal *et al.*, 1996), modelling performance was operationalized as difference against a reference solution prepared by the researchers. Results indicate that subjects performed better on the process task given the process tool, but for the object-oriented task, the tool (whether process oriented or object-oriented) was not a significant factor. Agarwal *et al.* (1999) test model interpretation by assessing comprehensibility of object-oriented and process models.

However, the model proposed by Vessey (1991) actually differs substantial from the proposal by Newell and Simon (1972) and make additional, implicit assumptions that cast doubt on the validity of the model for IS language research.

In (Newell and Simon, 1972) the task (external) environment is the one that the human problem solver is situated in. The problem space is the space in which problem solving takes place; it is an abstract space consisting of a goal description and symbols that can be manipulated by operators. "The effectiveness of a problem solving scheme depends wholly on its reflecting aspects of the structure of the task environment" and "the only aspects of the task environment that are relevant to solving a problem in a particular problem space are those that are reflected in the structure of that space." (Newell and Simon, 1972, p. 824).

On the other hand, Vessey (1991) introduces the notions of "problem representation", "problem solving task" which are external and an internal "mental representation" corresponding to Newell & Simon's problem space. Newell & Simon suggest that performance is a function of a match between the internal problem space and external problem space, while Vessey interprets this as suggesting that performance is a function of a match between the external problem space and the external goal definition.

This interpretation matches that of Newell & Simon only when the internalization of the external problem space is done in a bijective way and the match between the external space and the external goal remains in the internal representation. However, this is by no means an obvious conclusion as any external stimulus is processed and interpreted when internalized. Newell and Simon (1972) suggest that the problem space structure can be changed during problem solving, so that the mismatch problem may be solved as part of ordinary problem solving. A study by Kim and Lerch (1997) examines this in detail by viewing programming tasks as a search in different problem spaces. Their results support the idea that *mental, internal* representations are important in problem solving.

To summarize, the model by Newell and Simon (1972) cannot uncritically be assumed to be appropriate for situations where understanding and integration with pre-existing memory is of interest or where the interest focuses on match or mismatch between *external* problem definition and *external* problem space.

Cognitive fit has another shortcoming with regard to our aims. In the cognitive fit model, the task performance is the dependent variable that is of importance. In our suggested approach, the task performance is merely a *measure* of the true dependent variable, the domain understanding. Consequently, there is no place for "understanding" in the cognitive fit model because the task may or may not require understanding, i.e. the integration of the description with pre-existing cognitive structures and the ability to reason about the domain.

This is important since problem solving can be carried out without any understanding. Newell and Simon (1972) base their approach in computer science (see also Larkin and Simon, 1987) and draw an analogy from a Turing machine like their General Problem Solver (GPS) to humans. This problem solver operates directly on the internal representation without a need for understanding or integration of domain knowledge. The GPS manipulates symbols, but does not understand.

11.3 Experimental Design

Both field experiments and laboratory experiments are suitable ways to investigate the research question. A laboratory experiment has the advantage of control of the independent and extraneous variables and may thus rule out

many potential threats to internal validity. It does, however raise potential problems concerning external validity that a field experiment can address better. This is a trade-off for which a resolution must be found (Benbasat, 1989). In this case the most important threat to internal validity comes from subjects themselves and the increased control is more important than external validity. Hence, we employ a laboratory experiment. External validity is gained again through the results of the case study (Chapter 10).

11.3.1 Independent Variables

Since there is a large number of rules and corollaries, empirical verification of each rule and corollary individually is not feasible. Instead, the effect of different diagrams wich vary in their conformance to the proposed rules will be examined. These diagram types include diagrams created without semantic guidance that violate a large number of rules. A second type will be a diagram which adheres completely to the proposed rules. A third diagram type will violate a small number of specific rules. This set of diagram types covers a large difference in the rules conformance as well as a small difference.

11.3.2 Dependent Variables

We make use of the experimental procedure developed previously by Gemino (1999). Gemino (1999); Gemino and Wand (2001) use Mayer's model of learning (Mayer, 1989). This model suggests that learning outcome is determined by the learning processes, which in turn depends on the learning material, the instructional method and the learner characteristics. The learning processes are posited to involve encoding of material from short-term memory into long-term memory and the organization of material within short-term memory. Mayer (1989) does not detail how the memory is structured or how the encoding is done. His model is complementary to the theory of conceptual networks and spreading activation and explained by that theory.

Gemino (1999); Gemino and Wand (2001) propose to measure domain understanding as the dependent variable, not recall or comprehension. This is in accordance to our understanding of the purpose of conceptual model, the transfer of knowledge, which is achieved once the interpreter has the same domain understanding as the model constructor. Gemino (1999) operationalizes domain understanding and learning performance as the ability

to solve problems in the domain. The instrument to measure this consists of a model and a set of open ended problem solving questions, questions that cannot be answered with the information given explicitly in the model but require deeper domain understanding. This understanding is supposed to be delivered by the way the domain is modelled. To ensure that subjects examine all aspects of the model, the instrument includes a set of diagram comprehension questions. These are also useful to ensure that the information content of different diagrams is equal: If there is no difference in subjects' diagram comprehension across different diagrams, these are confirmed as informationally equivalent. This operationalization has been employed successfully by Bodart *et al.* (2001) in the examination of optional properties in Entity-Relationship Diagrams.

The present research adopts this instrument and modifies it to fit the specific domains used here. It is expected that the number of correct answers to problem solving questions will be higher (mean) for subjects that interpreted diagrams conforming to the suggested rules and semantics than for subjects that interpreted diagrams not conforming to the suggested rules and semantics.

11.3.3 Control Variables

Additional information in one diagram over that in another can be a serious confounding variable and mask the effect of the rule conformance of the different diagrams. This is controlled by a set of comprehension questions as suggested by Gemino (1999) but also by directly measuring subjects' perception of useful information content post-test. A new instrument will be developed for this.

Information is only relevant if it is useful information. Not all additional information in a diagram is useful for the task and will influence subjects' performance on problem solving tasks. Hence, the perceived usefulness of the diagram for the problem solving tasks is assessed post-test. The instrument developed by Moore and Benbasat (1991) is used and adapted to the usefulness of models, instead of usefulness of technology. A similarly adapted usefulness instrument was adopted by Lim and Benbasat (2000) for their study of multimedia technologies.

A third possible confound is the ease of interpretation of a diagram. There may be some graphical variations which makes one diagram easier to read than others. In other words, for a model which conforms to ontological

rules, it may not be the rules, but the specific graphical representation and subsequent ease of interpretation which effects the difference in domain understanding. To control for this factor, we adopt the ease of use instrument items from Batra *et al.* (1990) and Moore and Benbasat (1991). The latter are adapted to the ease of interpretation of models as used in (Batra *et al.*, 1990; Gemino and Wand, 2001).

Yet another potential source of confounds and bias is the domain under study. It is conceivable that some domains are inherently simpler or more familiar to subjects. This can potentially obscure the effect of rule conformance of the diagram and is controlled for in two ways. First we measure subjects' familiarity with the domain as part of the post-test questionnaire. Second, we employ two different domains. This not only controls for potential biases but also lends additional external validity to our results.

Finally, the familiarity with and knowledge of UML is a potential confounding factor. As the diagrams depict UML models of a certain domain, it may be easier for subjects with better UML knowledge to interpret the diagrams more easily. However, this reduced effort in interpreting the diagrams should not result in better understanding, as the speed of interpretation and the quality of interpretation or learning are different aspects. Nonetheless, this variable is controlled for by using a post-test questionnaire.

Related to the previous influence factor is the effort which subjects expend on the task. It is conceivable that some subjects spend more effort on the interpretation and understanding of the diagram. This is in part mitigated by the inclusion of diagram comprehension questions which require the subject to at least visually scan all major areas of the diagram. However, the effort spent on the actual problem solving task may still differ. We control for the effort expended by measuring the time that subjects spend on the different tasks.

The diagram in Fig. 11.1 shows the experimental design with independent, dependent and control variables. This diagram should not be confused with a causal diagram as used in structural equations techniques such as LISREL or PLS. It merely shows independent and dependent variables together with potential confounds that will be controlled for.

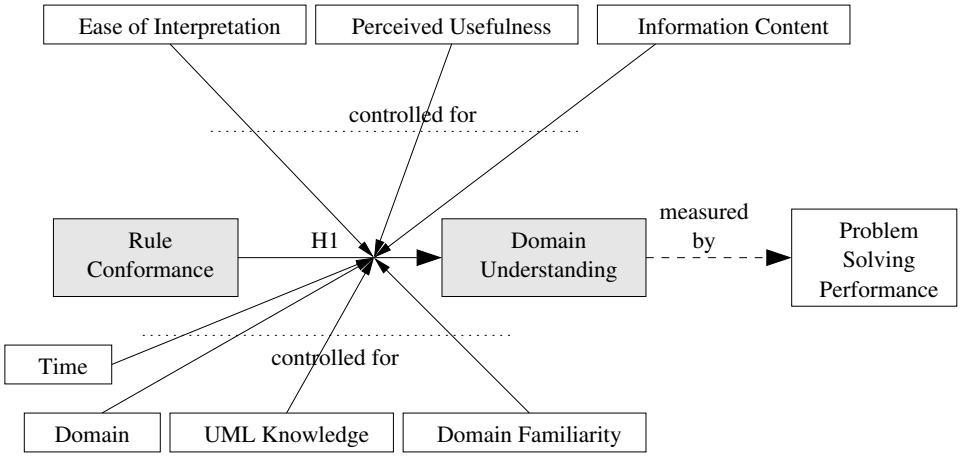


Figure 11.1: Experimental model showing dependent and independent variables with control variables

11.4 Instrument Development

Instrument development is done for the independent, dependent and control variable measurement instruments. For the independent variable, the diagrammatic models to be varied in the problem solving task have to be developed and the manipulation checked to ensure that diagrams do indeed differ in their conformance to the proposed rules.

The dependent variable is measured by a set of problem solving questions about the domain that is modelled. It must be ensured that these questions cannot be answered given the information in the model but that they are sufficiently close to the model so that variations in the model are relevant to the problem solving task. As there is no pre-determined set of correct answers, interpretation of the performance scores must be cross-validated by using multiple raters.

The control variables are assessed using a post-test questionnaire. Items for this questionnaire must be developed to ensure discriminant and convergent validity for all three control variables (Churchill, 1979; Hufnagel and Conca, 1994; Moore and Benbasat, 1991; Straub, 1989).

Independent Variable In this study, we investigate three diagrams which differ in their rule conformance in two domains. The first diagram (factor level "N") for each domain is taken from textbooks, to ensure that the modeller would use UML in a way that reflects current practice. Models are selected that are described by their author as reflecting the domain or as being analysis level models. Specifically, these models are not intended by their author as products of system or database design. The models selected are taken from (Fowler and Kendall, 2000) and from (Miller, 2002) and are discussed in Chapter 9 as Figs. 9.1 and 9.7 respectively. These models violate a large number of rules.

The models based on an ontological re-analysis of the two domains (Chapter 9) and constructed according to the proposed rules form the second experimental condition for each domain (Figs. 9.6, 9.12). These models conform to all applicable rules. However, they contain more model elements and thus their complexity is higher than that of the original models. This is factor level "R" of the independent variable.

While the complexity is not considered a direct influence on problem solving performance, it can influence ease of interpretation and more complex models may require more effort. Even though these influences are being controlled for, a third experimental condition was designed which is intended to mitigate the effect of complexity further. The models of Figs. 9.6, 9.12 are modified to violate a specific rule. All association classes are instead modelled as regular classes with associations to all classes which participated in the original association class. This violates rule 3 and as a consequence also rules 1 and 20. The models are shown in Figs. 11.2, 11.3 and form factor level "R2".

A manipulation check ensures that the intended variations of the independent variable have indeed the desired effect and that the experimental treatment conditions are different with respect to the independent variable. Three graduate students from computer science and MIS departments with good knowledge of UML were recruited to judge the models. All had used UML extensively in practical projects before. They were familiarized with the proposed rules and asked to assess the first two conditions for each domain.

The assessments for the initial models (Figs. 9.7, 9.1) ("N") indicated between 15 and 20 rule violations in each model. Some discrepancies arose as some judges disregarded repeated violation in their count while others did not. Judges were asked to build a list of all violated rules for each model

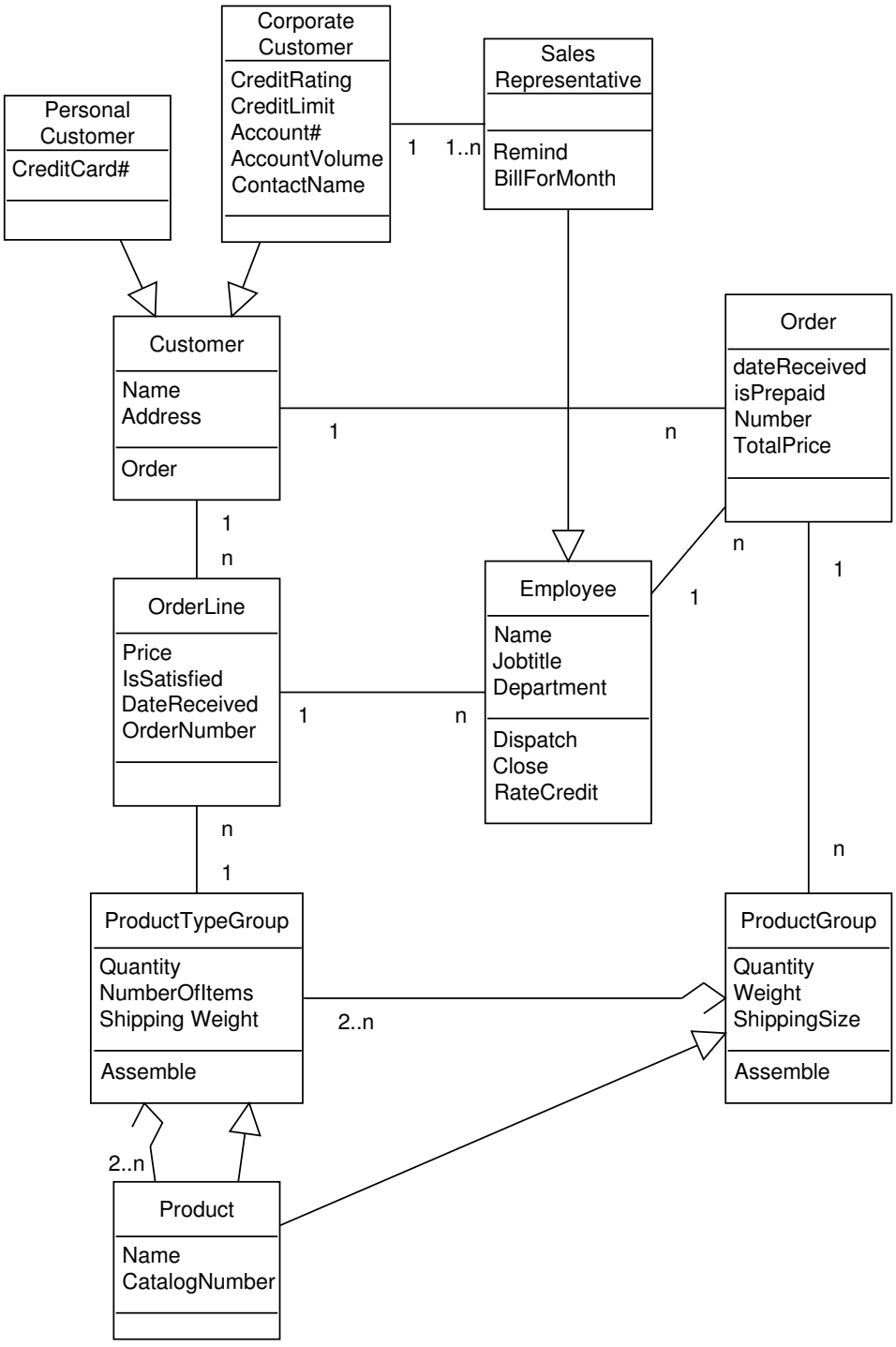


Figure 11.2: Third experimental condition, order processing domain

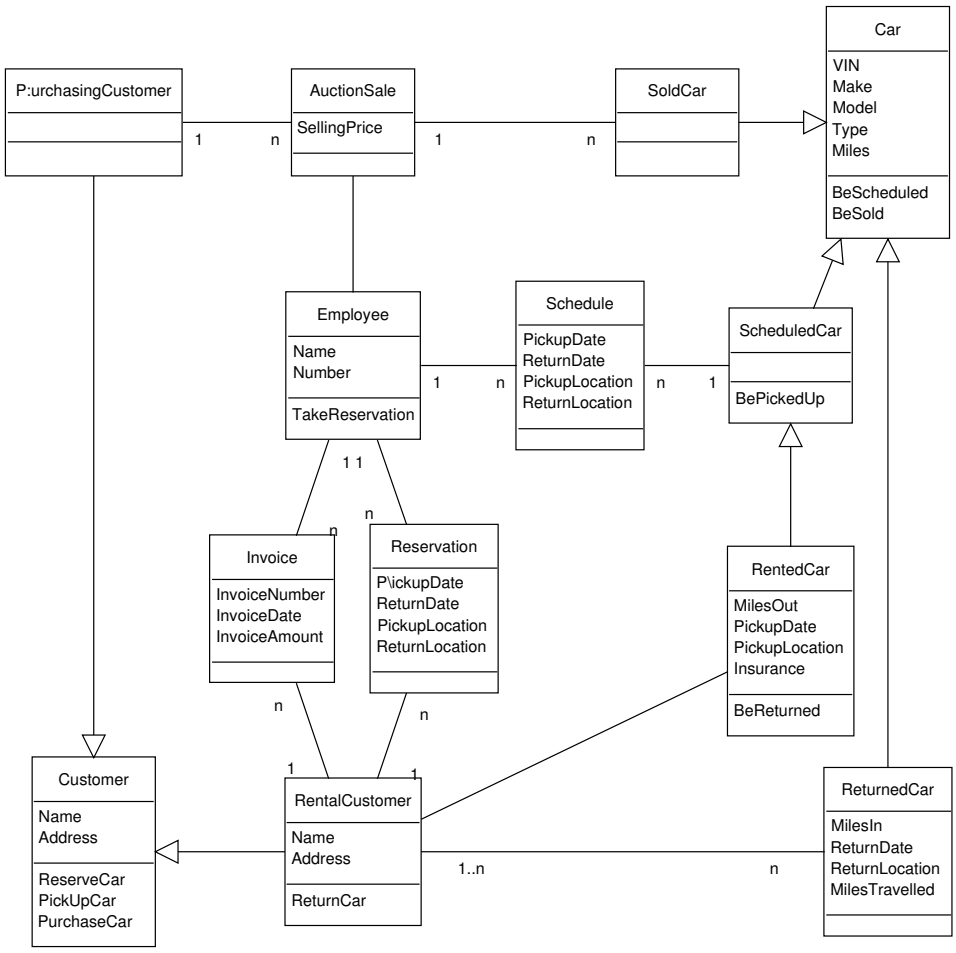


Figure 11.3: Third experimental condition, car rental domain

element. The lists of violations were found to be in good agreement. A formal analysis using inter-rater measures such as Cohen's Kappa was not performed. The main assessment was that the models violate a large number of rules and clearly did not conform to ontological semantics.

Similarly, the judges provided lists of violations and a count of violations for the models in Figs. 9.12 and 9.6 ("R"). The initial assessments showed some disagreements which were found to be rooted in misunderstandings of the model or certain rules. Upon further discussion these misunderstandings were cleared up and all three judges unanimously agreed that these two models conform to the applicable rules.

The final experimental condition was developed by automatically replacing each association class with a regular class that is associated with the participants of the original class ("R2"). No validation is necessary, as specific errors are introduced into each model in a tightly controlled way.

Dependent Variable The dependent variable, domain understanding, is operationalized as problem solving capability within the domain and measured by a set of open-ended problem solving questions (Bodart *et al.*, 2001; Gemino, 1999; Gemino and Wand, 2001). The problem solving questions (see Appendix I) were developed by using (Gemino, 1999) as a guideline. Care was taken that the questions cannot be answered directly from the information provided in any of the models. The questions were validated in two ways.

First, a graduate student in MIS examined the questions and the models to ensure that the information in the model is insufficient to answer the questions.

Second, a pilot test was conducted using a limited number of subjects (17) from the sample pool. Preliminary versions of the models were used. The task included diagram comprehension as well as problem solving. A wide variety of answers to the problem solving questions was provided by the subjects. Subjects provided correct as well as incorrect answers. The number of answers for a single question ranged from zero to eight and the answers given varied widely. Together, this indicates that no one correct answer can be deduced from the models.

While these techniques serve to validate the instrument to measure the dependent variable, the fact that the problem solving questions are open-ended questions implies the need to have the responses assessed by multiple

independent raters and cross-validated. This was done for the pilot-test and is done for the hypothesis testing using the generally accepted Kappa measure (Cohen, 1960) which measures inter-rater agreement of categorical responses (Nyerges *et al.*, 1998) (Sec. 11.8) below. The final set of problem solving questions used for the hypothesis test can be found in Appendix I.

Control Variables All control variables, except time to perform task, are measured using a post-test questionnaire. All control variables are measured by multiple items to ensure convergent validity of the instrument. The post-test items can be found in Appendix G.

To assess subjects' UML knowledge, they are asked to rate their UML knowledge on seven point Likert scale and to estimate the number of months for which they have used UML in actual projects, outside of the classroom environment ("UML-A", "UML-B"). These two items are not unproblematic (Hufnagel and Conca, 1994). The first measures subjects' own perceptions and interpretation of the extent of their knowledge. The most serious problem with this measure is the lack of inter-subject comparability. Two subjects that received the same kind of training in UML may rate their knowledge different, despite the fact that both would be able to solve the same problems. The second item can be fraught with recall problems and interpretation problems. One subject may interpret part-time experience as full-time experience. The resulting item scores will again lack inter-subject comparability. To address these shortcomings, an objective method is added. A set of 19 binary response (yes/no) and multiple choice questions relating to UML class diagrams are taken from Test Manager (2001), the test manager software to Hoffer *et al.* (2002), a popular systems analysis textbook ("UML-1" through "UML-19"). For purposes of analysis, the scores are added to form the variable "UML-TTL".

Domain knowledge or domain familiarity was also measured using a multi-item instrument. Subject's are asked to rate their self-assessed knowledge on a 7-point Likert scale ranging from low to high ("CR-1" for the car rental domain, "OP-1" for the order processing domain). Subjects were also asked whether they had worked in a car-rental company or order processing department ("CR-2", "OP-2") and so might have gained additional insight into the processes. While it is safe to assume that everyone would have ordered an item at one time or another, not everyone may have rented a car. Therefore we asked subjects whether they had done so previously or not ("CR-3"), as that can influence their domain familiarity.

The development of the instrument for measuring ease of interpretation, usefulness and information content follows the procedure outlined in Moore and Benbasat (1991). An initial item pool was built in part from existing instruments. The instruments found in (Moore and Benbasat, 1991; Gemino and Wand, 2001; Batra *et al.*, 1990) provided items that reflect the ease of interpretation variable. Items that reflect usefulness are taken from (Davis, 1989) as employed in (Lim and Benbasat, 2000). Both instruments are reported as highly reliable. An initial set of items reflecting information content was developed from discussions with MIS graduate students, as no previous instrument could be found in the literature. The set of items is listed in Appendix G.2. Items beginning with EOI are intended to reflect ease of interpretation, those beginning with USE are intended to reflect usefulness and those beginning with INFO are intended to reflect information content. Items "EOI-4", "EOI-6" and "EOI-9" are reverse coded items.

In order to ensure reliability, discriminant and convergent validity of the instrument, two rounds of card-sorting were conducted. The first round used eight MIS graduate students as judges. These were given the initial item set printed on 5" x 7" index cards and were asked to sort them into categories of items that "go together" and then provide names for the categories they formed. Judges were instructed not to force items into a category and instead put it in an "others" category instead. A researcher was present to answer any questions about the procedure that the judges might have. It took judges between 10 and 20 minutes to sort the items. The number of categories formed varied from two to five and seven out of eight judges included a category for miscellaneous items they were unable to sort into other categories but which they explained were unable to form a category with other items that could not be sorted into an existing category either.

The following list is a list of *categories* that were named by judges. The following list has been sorted to make the groupings of similar items more visible and duplicates have been removed.

- Ease of Use
- User friendliness
- Interpretation and understanding
- Ease of understanding
- Experience with diagram interpretation
- Ease of use of the diagram
- Perceived diagram comprehensibility
- Ease of interpretation and understanding

- Diagram interpretation
- Usefulness
- Effect of information on answers
- Helpfulness for answering questions
- Effectiveness for answering questions
- Usefulness and Utility
- Helpfulness of diagram
- Usefulness of diagram
- How much the interpretation helps domain understanding
- Ease of interpretation and usefulness
- Representation of Domain
- Objective measurements
- Information provided
- Diagram properties
- Diagram quality
- Descriptions of the diagram
- Quality of information provided
- Perceived ease of interpretation of diagram
- How faithfully, closely the diagram represents the domain
- Problems with the diagram
- Personal perceptions of the diagram
- Overall evaluation, aesthetics

Moore and Benbasat (1991) recommend assessing inter-rater reliabilities even at this stage using Cohen's Kappa (Cohen, 1960). This is problematic for two reasons. While it provides a statistical measure of agreement, it may not reflect content validity. Two judges may group the same items together but these two groups reflect different categories. Only once categories with a common definition exist, does inter-rater reliability capture common understanding and therefore content validity. This is not a serious problem as the list of categories shows that clearly there exists common understanding as to the domains covered by the instrument. A second reason is the fact that subjects have varying numbers of categories. On examination of the results, this is the result of splitting the construct into sub-constructs. E.g. one judge provided two categories, "Helpfulness for answering questions" and "Effectiveness for answering questions" reflecting the same construct.

Instead of formally assessing inter-rater reliability at this stage, close examination of the card sorting results reveals that all eight judges created categories reflecting the intended ease of interpretation, the usefulness or

helpfulness of the diagram and the information captured in the diagram. Three categories were formed, "Ease of Interpretation" (EOI, category 1), "Usefulness for Task" (USE, category 2) and "Information Content" (INFO, category 3). The task relates to the problem solving tasks in the study, the dependent variable.

In the second round of card-sorting, we presented 15 judges with the three categories and the same initial set of items. These 15 judges include seven of the original eight judges and eight additional MIS graduate students. Judges were provided with the category labels and were asked to sort the items, printed on 5" x 7" index cards, into the categories. Judges were instructed not to force items into categories if they felt the item did not fit. Instead, judges were asked to put such items aside. For the following analysis of reliabilities, this category was labelled "Others" (category 4). Pairwise inter-rater reliabilities were computed using Cohen's Kappa (Cohen, 1960) using the procedure in (Baron and Li, 2001)[4]. The average reliability was 0.6636 with a minimum of .3857 and a maximum of .9371 (Appendix K.1). Nyerges *et al.* (1998) give the following interpretation of this parameter:

Kappa	**Interpretation**
< 0.4	poor agreement
0.4 - 0.8	moderate agreement
> 0.8	Good to perfect agreement

Moderate agreement at this stage requires refinement of the items. Two kinds of items were excluded at this stage:

- Items that showed great variability in the categorization among judges. These items are ambiguous with respect to the underlying construct that they reflect. Different judges interpret these items as reflecting different constructs. Items "EOI-5", "INFO-3", "INFO-5" fall into this class of items.

- Items that were rated as category four, "others". These are items that the judges felt did not reflect any of the given constructs. Items "USE-1", "EOI-4", and "EOI-12" fall into this class of items.

Items that were found to be ambiguous with respect to the construct they reflect by some judges were also rated as not reflecting any construct by other

[4]All statistical analyses for this research were conducted using the statistical software R (Version 1.70) (Ihaka and Gentleman, 1996).

Construct	Items
Ease of Interpretation	EOI-1, EOI-2, EOI-3, EOI-6, EOI-7, EOI-8, EOI-9, EOI-10, EOI-11
Usefulness	USE-2, USE-3, USE-4, USE-5, USE-6
Information Content	INFO-1, INFO-2, INFO-4

Table 11.1: Item categorization after second round of cardsorting

Concept	α
Ease of Interpretation	.9366
Usefulness	.9121
Information Content	.5394

Table 11.2: Scale reliabilities determined by Pilot-Test (Cronbach α)

judges, whereas the assessment that items did not reflect any construct was universal across judges. Table 11.1 shows the categorization of the items after this step.

All three control variables are satisfactorily reflected by three or more items (Moore and Benbasat, 1991; Straub, 1989). The Kappa values for this set of items were assessed across judges and found to average .8294 with a minimum of .4769 and maximum of 1.000 (Appendix K.2). This shows good to excellent agreement across judges.

11.5 Pilot Test

In the next step, this instrument was pilot-tested. The primary purpose is to assess the convergent and discriminant validity of the Ease of Interpretation, Usefulness and Information Content scales. The pilot test used 14 subjects from the sample frame and was conducted using preliminary versions of the UML models. Each subject was presented with two different domains and a post-test was administered after each domain. For the purposes of assessing the reliability and validity of the control instrument the two observations for each subject are treated as independent, yielding a total of 28 observations. Scale reliability was assessed using Cronbach's α coefficient (see Appendix K.3). The results are shown in Table 11.2.

The reliabilities are excellent for the Ease of Interpretation ($\alpha = .9366$) and Usefulness ($\alpha = .9121$) constructs and adequate to good for the Information Content construct ($\alpha = .5394$). The result for the Information Content is likely caused by the fact that the last construct is measured on a three item scale, which is generally accepted as the lower bound for the number of items per construct[5].

A confirmatory factor analysis (maximum likelihood extraction with varimax rotation) was conducted to assess the convergent and discriminant validity of the scales (Appendix K.3). Due to the very small sample size, the samples of the two domains were combined to yield $n = 28$ observations. Hence, the observations are not completely independent, even though subjects responded to two different diagrams. For this reason, and the fact that even $n = 28$ observations is a small sample size, the following analysis cannot provide firm statistical inferences and are intended to be indicative only.

The rotated factor loadings are shown in Table 11.3. The variance explained by all three factors is 0.687. A χ^2 test of the number of factors required showed a non-significant p-value of .518 for a test statistic of 86.81 with 88 degrees of freedom. Following the procedure of Harman (1976), a two-factor solution was found to have a p-value of .156 while the single-factor solution had a p-value of .001. Statistically this would indicate a single factor solution. However, Harman (1976) warns that the χ^2 test yields too low an estimate for the number of factors on small data sets. He also cautions against the sole use of statistical arguments for the number of factors.

Table 11.3 shows clearly the "usefulness" construct as the first factor with good convergent validity (high loadings of items on that factor, $> .65$) and discriminant validity (low cross-loadings of items onto other factors, $< .35$). The other two constructs are problematic. Items "EOI-7", "EOI-8" and "EOI-10" form a clear and distinct factors, but items "EOI-2", EOI-3", and "EOI-6" cross-load heavily onto a third factor. Items "EOI-1" and "EOI-9" load onto this third factor altogether. This would suggest that Ease of Interpretation is a multi-dimensional construct. Similarly, the items intended to reflect Information Content do not clearly form a single factor.

[5]The α value for the scale reliabilties using the responses for the first domain (the first set of 14 observations) is .0360 while, when using the responses for the second domain (the second set of 14 observations, from the same subjects), it is .7286. This seems to suggest differences in the two domains which affect subjects' interpretation of the post-test questionnaire items. This effect will be revisited when analyzing the final data.

Item	Factor1	Factor2	Factor3
USE-2	0.747	0.244	0.142
USE-3	0.634	0.236	0.221
USE-4	0.813	0.164	0.211
USE-5	0.872	0.323	
USE-6	0.806	0.281	
EOI-7	0.357	0.837	0.130
EOI-8	0.382	0.780	0.403
EOI-10	0.432	0.749	0.397
EOI-3	0.334	0.630	0.649
EOI-2	0.244	0.622	0.690
EOI-6		0.529	0.462
EOI-11	0.481	0.382	0.264
EOI-1	0.406	0.244	0.678
EOI-9	0.220	0.233	0.628
INFO-1	0.505	0.530	
INFO-2	0.328	0.720	
INFO-4	0.129		-0.598

Table 11.3: Rotated factor loadings, pilot-test (values $< .1$ omitted)

However, due to the very small sample size, no firm statistical inferences can be drawn. A final confirmatory factor analysis is performed on the larger data set from the main study (see Sec. 11.8).

11.6 Subjects

The target population for this study is the set of business analysts participating in the analysis stage of an IS project. These are distinct from system analysts in that they attempt to understand the business, not the IS. Typically business analysts will have knowledge both of the business and the information systems aspect, although the business knowledge is not usually as deep as that of end users nor is the IS knowledge as deep as that of system analysts or system designers.

The sample frame for this study comprises undergraduate students in their fourth year. The subjects were drawn from three groups of students,

forming three groups of subjects. This is controlled for by including this variable ("Group") as a factor in the following analysis. One group of subjects has been drawn from a course in system analysis offered at the business school (factor level "3") while the others have been drawn from two sections of a system analysis course offered at the department of computer science. The latter two groups were enrolled in the spring section (factor level "C") and the summer section (factor level "C2") of this course. Both sections were taught by the same instructor and subjects from both sections participated in the study after the same amount of material had been covered in either section.

All subjects had been exposed to the UML syntax through the systems analysis course materials and lectures. Student's knowledge of UML and experience in applying UML are two control variables in the following analysis ("UML.TTL", "UML.A"). Student's participation in the study was voluntary and entirely anonymous. Students did not receive course credit for participation nor was credit withheld if students chose not to participate. The rationale for the study, theory and findings were made available to students after participating in the study. Subjects performed the experimental task in small groups of 5 to 6 students each, located in a small conference room to reduce any distractions.

11.7 Design and Procedure

The experimental design included a within-subjects factor as it was felt that the number of subjects that could be gained would not be very large. The factor "Domain" with the levels "CR" (car rental) and "OP" (order processing) was chosen as the within-subjects factor. This ensures minimal carry-over and learning effects, which would have been possible had the factor "Rules" (rule conformance of diagrams) been chosen as within-factor.

Subjects were randomly assigned to one of three diagram types differing by rule conformance (factor "Rules" with levels "N", "R", and "R2"). Each subject was given a questionnaire which included two diagrams and associated comprehension and problem solving questions. The two diagrams depicted the two domains ("Domain") and the order of the diagrams was varied randomly between subjects to control for and exclude potential biases from carry-over or learning effects. Subjects were asked to record the current time on various pages on the questionnaire. A wall clock was provided for

this purpose. Subjects were instructed not to revisit sections of the questionnaire once they were completed. For each domain, the questionnaire included post-test questions measuring the control variables pertaining to each diagram or domain. At the end of the questionnaire were the questions measuring UML knowledge.

11.8 Results

Once data was collected, the scale reliabilities for the control variables were checked again (Sec. 11.8.1) and the inter-rater reliability for the coding of the problem solving responses was assessed (Sec. 11.8.2). Then, a confirmatory factor analysis was conducted on the items for the control variables (Sec. 11.8.3). Finally, an ANCOVA procedure and a Linear Mixed Effects Model (LME) was used to test the hypotheses (Sec. 11.8.4).

11.8.1 Scale Reliabilities

The scale reliabilities for the control variables Ease of Interpretation, Usefulness, and Information Content were assessed using Cronbach's α measure. Table 11.4 shows the values for the α coefficient, for $n = 53$ responses for each domain. The first columns shows the values for the Car-Rental domain, the second column shows the values for the Order-Processing domain. This analysis confirms excellent scale reliabilities for the Ease of Interpretation and Usefulness measures with α measures greater than .9, while the Information Content measures still show adequate to good reliabilities ($alpha \approx .65$). This confirms the reliabilities assessed by the pilot-test (Sec. 11.5). Again, the lower result for the Information Content is likely a result of it being a 3-item scale. With this full data set, the values for both domains are very similar. Hence, the differences between the two domains exhibited in the pilot test (Sec. 11.5) are likely a consequence of small sample size.

11.8.2 Interrater Reliabilities

The problem solving questions measuring the dependent variable (see Appendix I) were open ended questions; subjects were asked to provide as many answers as they could think of. Therefore, no a-priori list of correct answers was available against which to check the given answers. Two graduate stu-

Domain	CR	OP
Ease of Interpretation	.9456	.9234
Usefulness	.9176	.9243
Information Content	.6297	.6897

Table 11.4: Scale reliabilities determined by Pilot-Test (Cronbach α)

dents with knowledge of the two domains were asked to rate the answers as correct or incorrect and to assess the total number of correct answers for each question for each subject.

Initially, the two raters each received 15 completed questionnaires and were asked to compile a list of correct answers. These lists were exchanged between the two raters who then combined their lists of correct answers. They each revisited the completed questionnaires they had rated to reflect any differences in assessment due to the combination of correct answer sets. They then independently rated the remainder of the questionnaires.

The final inter-rater agreement over all subjects was assessed using Cohen's Kappa (Appendix K.4). The agreement was computed for the number of correct answers that each rater assessed for each problem solving question (typically ranging from 0 to 5)[6]. The average Kappa value for 0.9138 with a minimum of 0.7872 and a maximum of 1.000 showing excellent agreement between the two raters (Nyerges *et al.* (1998)). For further analysis, the average of the two raters' scores was used (variable "Prob").

11.8.3 Convergent and Discriminant Validity

Convergent and discriminant validity was assessed using confirmatory factor analysis (maximum likelihood extraction with varimax orthogonal rotation) by computing a 3-factor solution. The rotated factor loadings for each domain are shown in tables 11.5 and 11.6.

For the car rental domain, the variance explained by three factors is .686. The table clearly shows the Usefulness construct as a single factor with very little cross-loading of items onto other factors. However, neither the Ease

[6] A more exact method would rate the assessment of both raters for each answer given, instead of the number of correct answers. While it is possible that by rating the number of correct answers disagreements between raters could cancel out, the sharing of lists of correct answers ensures adequate agreement.

Item	Factor1	Factor2	Factor3
CR-EOI-1	0.792	0.213	0.424
CR-EOI-2	0.755	0.279	0.488
CR-EOI-3	0.767	0.174	0.533
CR-EOI-6	0.680	0.101	
CR-EOI-7	0.490	0.206	0.765
CR-EOI-8	0.411	0.199	0.692
CR-EOI-9	0.662		0.168
CR-EOI-10	0.632	0.306	0.575
CR-EOI-11	0.613	0.176	0.372
CR-USE-3	0.343	0.796	0.183
CR-USE-4	0.355	0.810	
CR-USE-5	0.121	0.726	0.433
CR-USE-6	0.350	0.795	0.108
CR-USE-2	0.315	0.563	0.586
CR-INFO-2	0.168	0.605	0.657
CR-INFO-1	0.360	0.316	0.201
CR-INFO-4	0.164	0.395	

Table 11.5: Rotated factor loadings, car rental domain (values $< .1$ omitted)

of Interpretation nor the Information Content factor is clearly defined. The items intended to reflect Ease of Interpretation appear to express two factors, with items "CR-EOI-7" and "CR-EOI-8" heavily cross-loading. The items intended to reflect Information Content ("CR-INFO-1", "CR-INFO-2" and "CR-INFO-4") do not appear to show any coherent factor structure.

A very similar result is obtained for the order processing domain. Here, the Ease of Information items again load heavily and even more clearly on two distinct factors ("OP-EOI-7", "OP-EOI-8", "OP-EOI-10"). The Usefulness items clearly load on a single factor, again with very little cross-loading, while the Information Content items do not reflect any clear factor structure.

Generally, the Ease of Interpretation and Usefulness constructs are adequately represented by the developed item set. The main problem is that the Information Content measure does not show convergent validity. The Ease of Interpretation construct also exhibits item loadings that are indicative of two sub-factors, with items EOI-1, EOI-2, EOI-3, EOI-6, EOI-9, and

	Factor1	Factor2	Factor3
OP-USE-2	0.720	0.206	0.215
OP-USE-3	0.784	0.335	0.295
OP-USE-4	0.764	0.155	
OP-USE-5	0.876	0.185	0.201
OP-USE-6	0.894	0.174	
OP-INFO-2	0.587	0.336	0.245
OP-INFO-1	0.411	0.266	0.254
OP-INFO-4	0.382		
OP-EOI-1	0.154	0.822	0.299
OP-EOI-2	0.258	0.839	0.286
OP-EOI-3	0.208	0.772	0.381
OP-EOI-6		0.547	0.173
OP-EOI-9	0.111	0.600	
OP-EOI-11	0.346	0.522	0.389
OP-EOI-7	0.256	0.384	0.885
OP-EOI-8	0.195	0.448	0.653
OP-EOI-10	0.291	0.443	0.657

Table 11.6: Rotated factor loadings, order processing domain (values $< .1$ omitted)

EOI-11 forming one factor while items EOI-7, EOI-8, and EOI-10 forming another factor. Referring to the items making up this factor, it appears possible that there is a difference between the ease of understanding a diagram and the ease of interpreting a diagram.

However, what is of primary interest for this study is to control for usefulness (Sec. 11.9): While additional information may or may not be present, it is not relevant if not found useful for the task. Bartlett factor scores are computed for further use of these factors as covariates for the linear models used in the next section (variables "Interpretation", "Usefulness", and "Information").

11.8.4 Hypothesis Testing

Different statistical techniques, such as ANOVA or regresseion analysis, can be brought to bear on the model shown in Fig. 11.1. Due to the variables rule conformance and group being 3-level factors, a t-test for difference of means cannot be used. While a regression analysis would allow the estimation and significance tests of the continuous control variables, such as ease of interpretation and usefulness, the aim of this study is to identify possible differences in problem solving behaviour due to the different levels of rule conformance. All other variables are considered control variables, and are not of primary interest in this study. Due to this reason, an analysis of variance with continuous co-variates is appropriate. As the experimental model includes a within-subject component, a repeated measures, mixed-effects ANCOVA is appropriate (Shavelson, 1996). The null-hypothesis of this study is that the means of problem solving behaviour across all three levels of rule conformance are identical. Rejection of this hypothesis at the level of significance generally accepted (.05) then indicates that rule conformance does indeed lead to differences in problem solving capabilities. Control variables are included in order to explain variance in the data, but their parameters are not of interest, so that a regression analysis is not required. Further, as the means in Tab. 11.7 clearly show the differences, a Tukey (1977) or Scheffe (1959) post-hoc test for specific means is not necessary.

While the chosen ANCOVA approach is suitable also for within-subject experimental designs as in the present case, the more recent linear mixed effects (LME) approach (Pinheiro and Bates, 2000) can complement this technique. In contrast to ANOVA and ANCOVA techniques, LME does

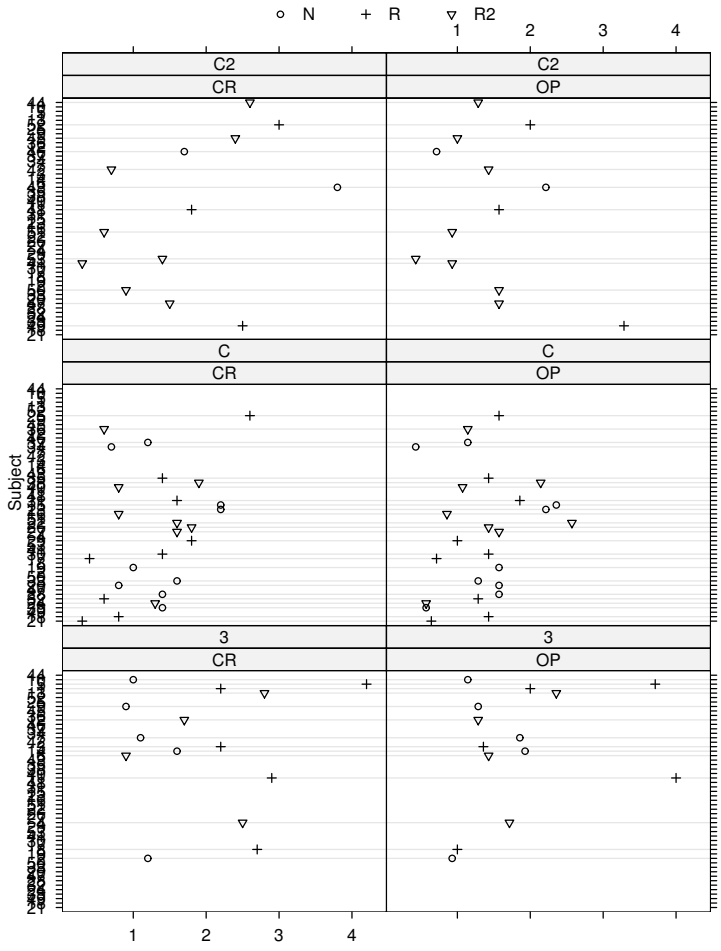


Figure 11.4: Scatterplots by group and domain

not maximize the explained variance of the model, but instead computes a maximum likelihood estimate of the model parameters, i.e. the group means of the factors and regression coefficients fo the covariates. The LME approach provides ultimately the same information as the ANCOVA, but uses a different statistical method. Its use in this research serves to provide additional validation of the results, a multi-method approach to data analyis intended to increase the robustness of the statistical results.

Further, for models involving a larger number of fixed and random effects, as is the case in the present model, the linear mixed effects modelling technique allows more expressive modelling. The advantage of LME modelling is the ability to expressly account for random effects such as experimental subjects in the present within-subjects design, and to model nested structures of such blocks. LME can also help deal with asphericity of data, non-uniformity of errors across factor level, as potential sources of asphericity can be included as random effects. The main text of this thesis provides the standard ANCOVA approach while the LME approach can be found in Appendix J.3. Post-hoc validation of model assumptions for both ANCOVA and LME can be found in Appendix. J.2.

	Rule Conformance				**Domain**			**Subject Group**			
	No Rules (N)	Rules (R)	Partial Rules (R2)	Results	Car Rental (CR)	Order Proc. (OP)	Results	Business Students (3)	Comp. Science Students 1 (C)	Comp. Science Students 2 (C2)	Results
	Means (SD) n=16	Means (SD) n=17	Means (SD) n=20	F (sig.)	Means (SD) n=53	Means (SD) n=53	F (sig.)	Means (SD) n=14	Means (SD) n=26	Means (SD) n=13	F (sig.)
Problem Solving Performance	1.4558 (0.6720)	1.8437 (1.003)	1.3996 (0.6439)	4.0583 (.021) *	1.6019 (0.8648)	1.5162 (0.7388)	0.4422 (.508)	1.9250 (0.9406)	1.3313 (0.5736)	1.6203 (0.9002)	6.687 (.002) **

Table 11.7: Main effects of rule conformance, domain and subject group on problem solving performance, measured in avg. number of correct answers per question

Before any statistical methods are applied, the data is examined visually and summarized in Table 11.7. Note that Ease of Interpretation, Usefulness and Information Content are derived factor scores. These are by definition zero-centered. Since factor scores were computed independently for each domain, there is no difference in the mean of these variables between the two domains.

Fig. 11.4 shows all data points plotted against subjects with indications of the rule conformance of the diagram ("N", "R", "R2"). The observations are grouped by domain and subject group. No general tendencies or abnormalities can be visually identified.

An initial plot of the data shows the main effects of and interaction effects between the major factors of the model. Fig. 11.5 shows box and whisker plots (Tukey, 1977) of the following main effects. In the top diagram, it shows the main effect of rule conformance of the diagram ("Rules") on problem solving performance; in the center diagram it shows the main effect of the domain ("CR" = car rental, "OP" = order processing) on problem solving performance; in the bottom diagram it shows the main effect of the group of subjects ("3" = Business undergraduates, "C", "C2" = first and second group of computer science students) on problem solving performance. The domain does not appear to have an effect. Fig. 11.6 shows the interaction effect of rule conformance ("Rules") and the domain on problem solving performance, while Fig. 11.7 shows the diagram and group interactions.

These plots suggest that the domain may not be relevant in the statistical model as the two domains follow the same pattern with respect to problem solving performance by rule conformance. The difference in problem solving performance between the two domains is very small, compared to the effect size of rule conformance. However, the second interaction plot (Fig. 11.7) shows that the first group of computer science subjects did not follow the general pattern of both the business students and the second group of computer science students. That pattern indicates an increase in problem solving performance for the diagram which conforms to all rules, while the problem solving performance for the two other types of diagram appears lower.

These visually suggested effects are analyzed statistically by the ANCOVA procedure and Linear Mixed Effects (LME) modelling[7]. The fol-

[7]The statistical analysis is done using the `R` statistics package with the procedure shown

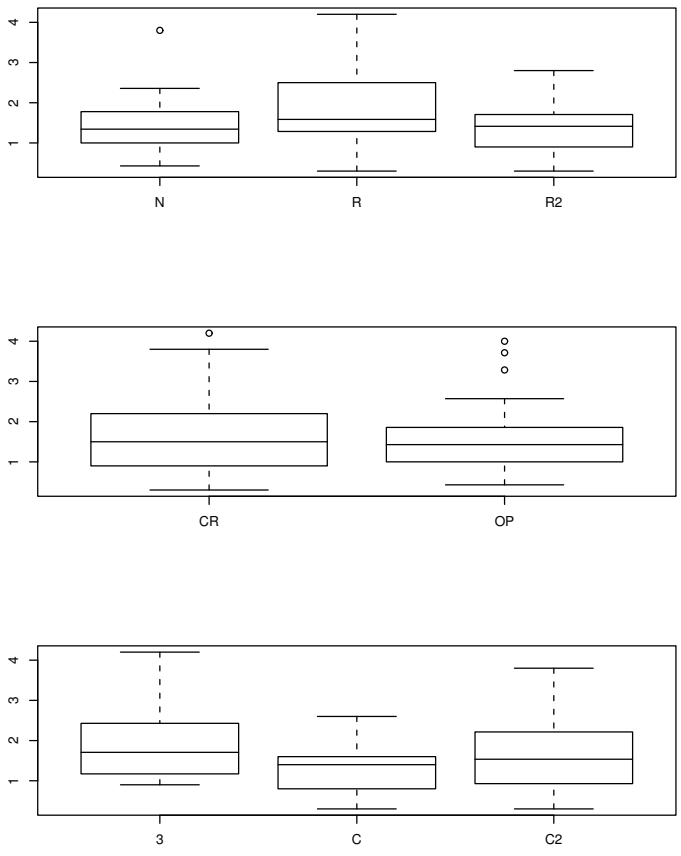


Figure 11.5: Box plots showing main effects of Rules (top), Domain (middle) and Group (bottom) on problem solving performance.

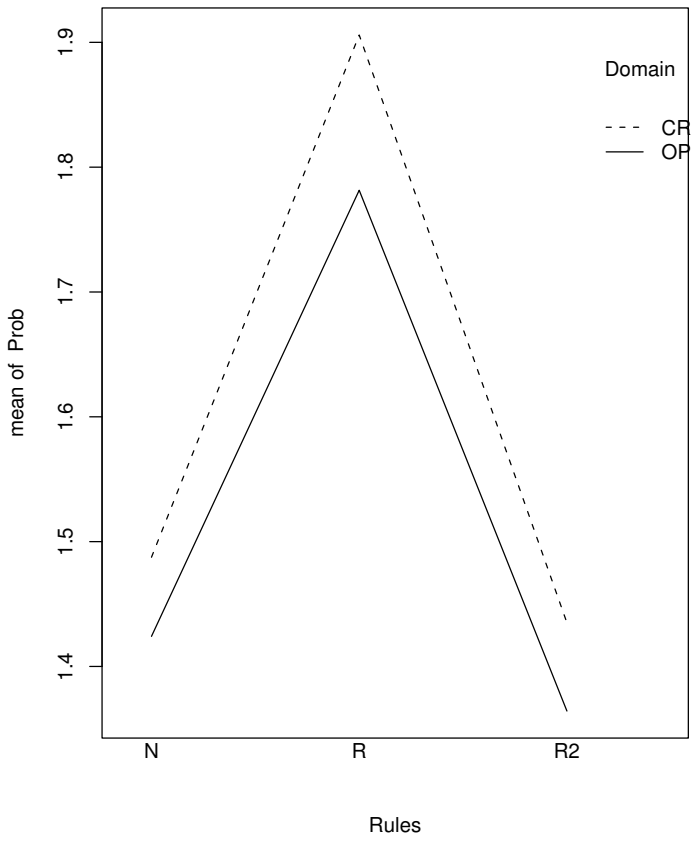


Figure 11.6: Interaction effects of Rules and Domain on problem solving performance

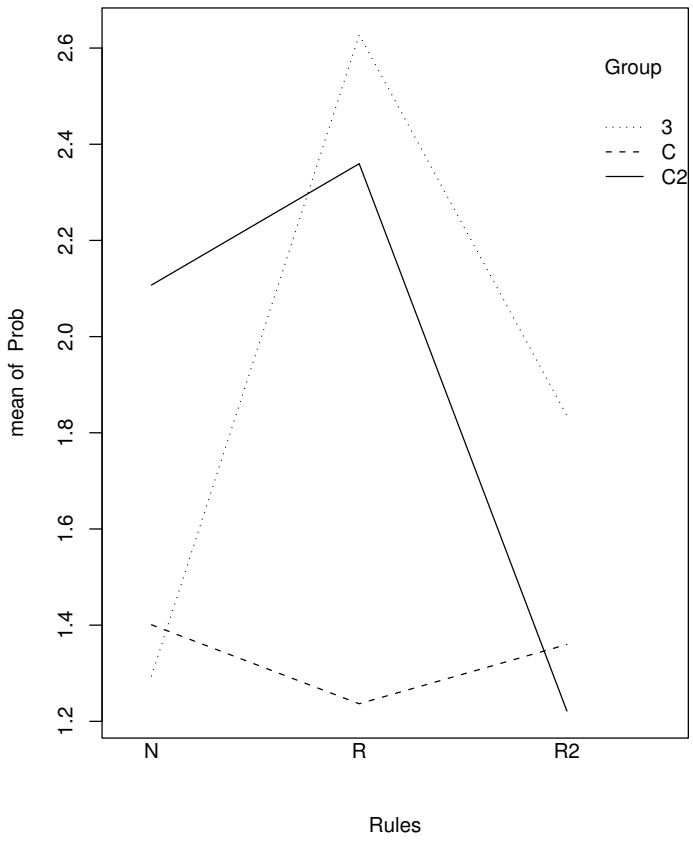


Figure 11.7: Interaction effects of Rules and Group on problem solving performance

lowing paragraphs show the ANCOVA analysis. The LME analysis can be found in Appendix J.2 and confirms the ANCOVA results.

ANCOVA The ANCOVA model is set up to include the factors "Rules", "Group" and "Domain" as well as their first order interactions and further includes the following control variables as continuous covariates without interaction effects:

- UML.TTL: Total score on the UML assessment questions
- UML.A: Self-assessed UML knowledge (7-Point Likert)
- SelfAsssess : Self-assessed domain knowledge (7-point Likert)
- Time: Time taken for problem solving
- Comp: Average comprehension score
- Interpretation: Ease of interpretation factor scores
- Usefulness: Usefulness factor scores
- Information: Information content factor scores

Table 11.7 shows the results of the ANOVA for the main effects of "Rules", "Group" and "Domain". There is a statistically significant difference in the means between different levels of rule conformance ($p = .21$) confirming the visual identification of the effect. The mean for the fully rule conforming diagrams was 1.8437 compared to 1.4558 and 1.3996 for the non-conforming and partially conforming diagram. This indicates a statistically significant increase in problem solving performance. The domain did not have a statistically significant effect, with means of 1.60 and 1.51 for the two domains ($p = .51$). There was a main effect of group on problem solving performance (p=.002).

The full results below also indicate a highly significant interaction effect of "Rules" and "Group" (p=.002), which confirms the effect observed visually in Fig. 11.7. This will be discussed further below.

```
Error: Subject
```

in Appendix K.7.

```
      Df Sum Sq Mean Sq
Rules  1 1.3804  1.3804

Error: Within
               Df Sum Sq Mean Sq F value   Pr(>F)
Rules           2  3.574   1.787  4.0583 0.020819 *
Domain          1  0.195   0.195  0.4422 0.507916
Group           2  5.889   2.944  6.6868 0.002030 **
UML.TTL         1  5.002   5.002 11.3609 0.001140 **
UML.A           1  0.006   0.006  0.0145 0.904564
SelfAssess      1  1.040   1.040  2.3624 0.128095
Time            1  1.035   1.035  2.3507 0.129029
Comp            1  1.215   1.215  2.7591 0.100473
Interpretation  1  0.021   0.021  0.0485 0.826308
Usefulness      1  0.352   0.352  0.8000 0.373691
Information     1  1.054   1.054  2.3934 0.125655
Rules:Domain    2  0.355   0.178  0.4032 0.669475
Rules:Group     4  8.157   2.039  4.6314 0.002004 **
Domain:Group    2  1.646   0.823  1.8692 0.160683
Residuals      83 36.546   0.440
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

The goodness of fit of the model is assessed as the ratio of explained versus total variance. This ratio is $R^2 = .4582$ which suggests that the explanatory power of the model is adequate[8]. The data also shows that a number of covariates do not have any explanatory power (low Sum Sq) and are not statistically significant.

The estimated improvement in problem solving performance for the rule conforming diagrams is 0.388 (average correct answers per question). This represents a 26% increase in problem solving performance. The performance for the similar diagrams violating only one rule decreased by about 0.056 (average correct answers per question) but this effect is not statistically significant. While the effect of total UML knowledge (UML.TTL) is statistically significant, the small magnitude of the parameter (.089) makes it doubtful that it is practically relevant.

[8] R^2 ranges from 0 to 1. An $R^2 > .7$ is considered excellent.

A further, extended, analysis using ANCOVA techniques and Linear Mixed Effects Modelling can be found in Appendix J. A more parsimonious model was derived using step-wise inclusion of variables (Appendix J.1. The LME analysis is shown in Appendix J.2 and confirms the ANCOVA results in this chapter. A linear mixed effects allows more explicit modelling including fixed and random factors, such as occur in this within-subjects design. Model assumptions of ANOVA and the LME model are tested in Appendix J.3 and found to hold for the data.

Equivalence of Diagrams One of the main assumptions is that the models must be informationally equivalent with respect to information relevant to the task. Otherwise, the additional information could lead to improved problem solving performance. This assumption is assessed by analyzing the effect of the different model types on the factors Information Content, Usefulness and Ease of Interpretation. This is done using an ANOVA procedure including "Domain", "Group" and "Rules" together with second order interactions as independent variables (Appendix K.8). Detailled results are shown in Appendix J.4.

The results shows that there appears to be an effect of the subject group on the perceived Information Content. However, this is uncertain due to the measurement difficulties identified above. In any case, this effect on Information Content does *not* appear to have an effect of Usefulness for problem solving. In fact, none of the three examined factors had any effect on Usefulness. There was a significant effect of rule conformance of Ease of Interpretation. The Ease of Interpretation for "R" diagrams ($\mu = -.4571$, $\sigma = 1.103$) was significantly less than that for "N" ($\mu = .3111$, $\sigma = 1.008$) or "R2" ($\mu = .1400$, $\sigma = .9517$) diagrams. Diagrams conforming to ontological semantics and rules are harder to interpret. However, ease of interpretation does *not* have a significant effect on problem sovling performance. This rules out an interpretation of Ease of Interpretation as a mediator variable.

In conclusion, the results of the present analysis justify the assumption of equivalence of the diagrams for the purposes of the problem solving task.

11.9 Discussion

Overall, the results confirm the hypothesis and underline the benefits of the ontologically derived modelling rules. Both the ANCOVA and LME

analyses identified the same significant influence variables, which are highly significant in the ANCOVA analysis and approached significance in the LME analysis. The detected effect sizes are not only statistically significant but large enough to be practically significant. The effect of modelling with ontological rules leads to approximately a 26% increase in problem solving performance.

Another important aspect indicated by the results is the lack of any significant difference between the performance on diagrams that completely lack ontological rules and diagrams that follow all but a few specific such rules. While this effect was only shown for the specific set of rules violated in condition R2 in this study (violating the representation of mutual properties by association class attributes), it can possibly point towards indications that the benefit derived from rule conformance may not be a linear phenomenon, i.e. that the benefit may only be realizable by a fully conformant model. At the very least, this finding indicates that there may exist certain threshold levels of rule conformance for certain rules or sets of rules. In terms of the underlying cognitive network theory, this implies that the entire model may have to match the mental network in order to be properly integrated.

It further appears that UML knowledge, as assessed by the comprehensive post-test questionnaire ("UML.TTL") also plays a partial role in explaining the derived benefits, although the estimated effect size ($\approx .09$) does not appear to be practically significant. The self-assessed UML knowledge ("UML.A") did not appear to have any impact, which suggests that it differs significantly from the objectively measured UML knowledge.

When examining possible differences in the Ease of Interpretation, Usefulness of the diagrams or the Information Content, only an effect on the Ease of Interpretation could be found. It appears that diagrams conforming to rules are easier to interpret than diagrams that do not conform to any of the rules. However, there is also a difference between the fully conformant diagram ("R") and the one which violates a few rules only ("R2"). This difference may stem from the lack of the ternary association symbol (diamond shape symbol). This reduction in the number of UML constructs used in the model may have influenced the Ease of Interpretation assessment. However, the original diagrams ("N") did not make use of this construct either, so this explanation does not tell the whole story.

Chapters 9 and 10 showed that following the proposed rules and adhering to the proposed ontological semantics tends to add information to

the model. It is possible, and indeed the analysis has shown, that this information makes the resulting models more difficult to interpret. However, this increased difficulty of interpretation has no impact on the domain understanding. Any claim that the rules should not be followed as they lead to more difficult to interpret models, fails to take into account the purpose of the model, as ultimately it must convey domain understanding, not necessarily be easy to interpret. Hence, the empirical results of this chapter confirm the argument made in Chaps. 9 and 10 that an initial diagram should be built which fully conforms to the proposed semantics. Only in a second, explicit, transformation step, may information be removed or rules violated to make the model e.g. better suited for database implementation.

The most interesting phenomenon is the interaction of the rule conformance of the diagrams with the group that subjects are drawn from. No outliers are present in the data which could account for this, suggesting that a deeper explanation is needed. One could have expected differences between the computer science subjects and business student subjects, because of their different educational history. Instead differences arose between two groups of computer science student subjects. This is all the more surprising as they had received the same instructions in UML and were taught by the same instructor. Subjects were recruited after the same stage in their UML instruction. Possible explanations are motivational. The second set of subjects were taken from a summer semester course and may have been more motivated to perform in the study. As the university that subjects were drawn from does not generally offer summer courses, it is possible that these were highly motivated students making use of the offering of that course.

A test for an influence of subject group on time spent on the task supports this explanation (Appendix K.8). Subjects in group "C" spent about 2.5 minutes less on the problems than the average. However, this did not translate into a significant effect of time on problem solving performance.

The effect of subject group on UML knowledge was also examined in search for an explanation. Both computer science student subject groups actually had significantly lower scores on the UML knowledge instrument than the group of business students. However, there was no significant statistical difference between the two, ruling out differences in UML knowledge as a possible explanation.

11.10 Potential Limitations

Generally, potential limitations of an empirical study can arise out of threats to internal, external or statistical validity. Internal validity may be compromised due to factors outside the model influencing the dependent variable. In order to minimize the potential for this, subjects were randomly assigned to a domain. While the study attempted to control for a number of potential factors that could influence the problem solving outcome, the model fit, which is only average, indicates that factors other than those examined here may play a role.

Another issue is the statistical power of the experiment. (Howell, 1997) suggests that in order to detect a medium effect size at the 0.05 significance level requires about 30 subjects. The size of the effect in our study was practically significant and with 53 subjects clearly detectable, the ANCOVA model was clearly significant and the same factors approached significance in the LME model.

A third potential issue is the measurement instrument itself. The instrument is derived from previous work (Gemino, 1999; Bodart *et al.*, 2001) and has been re-validated. One potential problem may stem from the fact that the convergent and discriminant validity for the Information Content control variable was marginal. However, the main confounding effect should not have come from different information content between diagrams but from differences in the perceived usefulness of the various diagrams. The usefulness scale on the other hand was derived from Moore and Benbasat (1991) and re-validation showed excellent validity.

External validity may be threatened by a potential lack of generalizability from subjects to target population. As outlined above, the target population profile roughly matches that of the experimental subjects, but any generalizations beyond this target population should only be made with great care. The results also clearly show differences in the way that different groups respond to the different diagrams. This is all the more difficult to explain as the difference was not between computer science students and business students but between the two groups of computer science students. Future research must aim to find a more homogeneous sample set which is also a better representation of the target population. Subjects should be participants in actual IS development projects in actual organizations. However, the resource demand required for such a study may make this infeasible.

Chapter 12

Contributions

This thesis has assigned ontological real-world semantics to UML. While it is not the first work to assign semantics to UML, it is the first to do so not by translating UML to other, more formal languages. Instead, we have mapped UML elements to elements of the real world, such as found in business and organizational domains. For example, things have been mapped to objects and mutual properties to attributes of association classes. Conversely, UML-events have been mapped to state transitions, etc.

This mapping can serve as the first step towards a commonly accepted meaning for UML constructs in business and organizational modelling and can contribute to a better understanding of conceptual models. As UML is an evolving language, this result is relevant from a theoretical perspective as it can guide that evolution of the language. For the practitioner, this creates a common language that business analysts can employ and with which IS and software designers are immediately familiar, helping to bridge the often wide gulf between the two. This enables all participants of an IS development project to realize the full potential of conceptual models as a communication medium in reasoning about the problem domain.

Furthermore, this is not the first study to examine UML from the perspective of the BWW-ontology. However, it is the first study to do this constructively. Previous studies such as (Opdahl and Henderson-Sellers, 2002; Dussart *et al.*, 2002) pointed out weaknesses by examining the mapping. This research goes beyond pointing out the ontological defects and instead suggests rules and guidelines for the modeller and analyst which alleviate these problems and enable the use of UML for conceptual modelling

of real-world domains. These rules not only show how certain aspects of the real-world domain are to be modelled in UML, but also guide the modeller to seek out and include additional information in the model. This latter effect is demonstrated by the two examples in Chap. 9 and the case study in Chap. 10.

For the academic community, this study shows a way to extend the idea of ontological evaluation in a more constructive way. It provides solutions to possible ontological defects found during an evaluation. For example, one could simply argue that operations and methods are ontologically redundant. Yet, the fact that they play a prominent role in object-oriented techniques makes it clear that a consistent way of modelling them must be found.

For the practitioner, the rules and guidelines developed are immediately accessible and can be put to practice. They are operationalizable and specific to enable their use in real modelling and analysis projects. The analysis of UML and derivation of rules provides a practical and detailed guide to using UML in business analysis. This is underscored by the fact that the majority of the developed rules can be expressed by the OCL language. This adds support for the modeller by reducing any ambiguitites.

The formal aspect of this research answers the call for more meta-model based research (Rosemann and Green, 2002; Davies *et al.*, 2003). Formalization serves to make the results unambiguous and operationalizable in CASE tools, so that automated support can be provided to the conceptual modeller. For the practitioner, such tool support may be useful in guiding the modelling process, rather than applying the rules to the final model and analyzing the result after the process. The case study shows this processs. The initial model by the project team was not known to the researcher and an independent model was developed by following the guidelines provided by the rules. The size and complexity of the models also shows that CASE tool support is useful for such models.

Formalization and the use of OCL therefore furthers the practical relevance of this investigation. Potential CASE tools incorporating the proposed semantics can be used to make this work easily accessible to practitioners without having to understand the underlying theory.

For the academic community, it shows the value and the usefulness of language meta-model for ontological analysis and the assignment of semantics. Using the meta-model, our results are also able to better guide the development of the language syntax to encompass conceptual modelling re-

quirements. As the official UML standard includes formal constraints in OCL on meta-model elements, it is possible to add the proposed rules and constraints to this set.

We have shown that the general process of our analysis is applicable to other languages. Rules relating e.g. to objects, attributes, and operations are generalizable to other object-oriented langauges. Some of the more specific rules and OCL expressions relate to idiosyncrasies of UML, e.g. the treatment of signal receptions, actions, and events.

However, this clearly shows that other languages cannot only be evaluated ontologically, but that prescriptive modelling rules can be derived from such an analysis. The methodology involving the transfer of ontological assumptions by virtue of a mapping is universally applicable to all languages. This study is a demonstration using UML as an example, as it is the most widely known object-oriented modelling language.

The two examples and the case study provided in this thesis serve to demonstrate the applicability of the results. The two examples show the difference between models with and without ontological semantics. It shows that UML models, even though labelled conceptual or analysis model by the modeller, in fact do not truthfully represent the real world. They carry hidden, implicit assumptions, a prime example of this is the alternative interpretation of the order processing example in Chapter 9. The examples and the case study show that models constructed with the proposed mapping and rules include more information than is needed for the IS design stage. It is this contextual information that can help understanding the domain. Also, both case study and the examples show that the level of detail is increased by following the proposed rules. For example, the initial model by the project team collapses a number of different attributes into a single class, while the independently developed model separates these and is therefore able to indicate semantic relationships between them.

While the difference between analysis and design models is widely acknowledged in the research community, the case study can provide an example of the differences. It also shows that the process of conceptual modelling using the provided semantics and rules does lead to useful outcomes. The developed models were of a size and complexity that is neither trivial nor beyond the capability of analysts to comprehend. It is a modelling process that is practicable and usable in real projects. For the practitioner, the two examples and the case study can show how to 'think ontologically'. They show a rigorous exclusion of any IS requirements or implementation concern

and focus strictly on real-world phenomena.

Finally, the experimental corroboration of the results cannot prove the proposed theoretical results to be true. It does however lend credibility to the rules and semantics. The results support the hypothesis that rule conformance leads to better domain understanding, a critical requirement in any IS project. Rule conformance had a statistically and practically significant effect on problem solving behaviour, which is used as a measure of domain understanding.

The experimental study builds on earlier research by Gemino (1999) who developed the instrument for assessing problem solving behaviour. This study successfully applied an adapted version of this instrument, thereby providing further support for the usefulness of this instrument in IS research. This study also employed an instrument developed by and adapted from Moore and Benbasat (1991). This instrument was revalidated to measure potential influences such as Ease of Interpretation, Usefulness and Information Content of the models. It provides further support for the usefulness of the original instrument.

To the practitioner, the experimental study complements the case study to demonstrate not only the applicability but also the benefit of the proposed rules and semantics. It shows that the purpose of a conceptual model, i.e. furthering domain understanding, is indeed supported by the developed theory. Problem solving performance using a rule conforming diagram was about 26% greater than problem solving performance using a non-conforming diagram.

Finally, this work serves as another pragmatic pillar of support for the BWW-ontology. As outlined in Chapter 2, as a fundamental philosophical assumption, the choice of the BWW-ontology cannot be debated or criticized a-priori and the end justifies the means. In this case, the end result does indeed justify the choice of ontology, as the empirical results, both case study and experimental study, indicate the usefulness and beneficial effects of the theory developed on the basis of the chosen ontology.

Chapter 13

Future Extensions

Future extensions to this work are possible both in theoretical deductive research as well as in the empirical research area. In a first step, the obtained formal results can be operationalized in an actual CASE tool. This can serve to further demonstrate their immediate applicability. This CASE tool can serve as the object of empirical evaluation. It must be shown how such an implementation of our results can benefit the modeller during the process of analysis and conceptual modelling. The fact that most CASE tools are based on language meta-models makes should facilitate the inclusion of the results.

In Chapter 1 the case for the use of UML as exemplar language was motivated by the fact that it would ease the transition from analysis to design. While the models developed according to the suggested rules and guidelines are valid UML models and therefore tranferrable to software and programming code, model transformations may still be necessary for a variety of reasons. Such transformations may for example increase the computational efficiency of the resulting softare or adapt the software design to a specific underlying database technology or programming language. While such implementation driven transformations are outside the scope of this thesis, they form an interesting area for further research.

While UML is the most widely accepted language for IS design, there exist other IS design languages, object-oriented and otherwise. The methodology of this thesis can be applied to the study of these languages as well. Besides providing real-world semantics to the target language, it would also support the suitability of the proposed method. More constructive analysis

of languages such as ER diagrams, OML and ARIS are needed that not only point out the incompatibilities with the BWW-ontology but show how these can be avoided.

The meta-model formalization in this work may be used as the starting point for a formal analysis. Techniques in the area of schema matching (Batini and Lenzerini, 1986; Rahm and Bernstein, 2001) appear useful to formally compare two meta-models. A meta-model of the BWW-ontology was developed by Rosemann and Green (2002). A similar, more complete, model can be developed in UML, which makes it comparable to the UML meta-model. Schema based techniques from schema matching research can then be applied. This provides a better basis than the mappings made in this thesis and in prior ontological evaluations. Those mappings, while intuitive, are based on the researchers understanding and interpretation of the language and the ontology. In this research, we have attempted to provide a comprehensive analysis to ensure that 'all the pieces fit', which at least to some degree alleviates this problem.

In the empirical research domain, the results of the experimental study pointed out possible future studies. The generally only adequate fit of the models with the data shows that there may exist further variables that have not been investigated. Future research might attempt to be explore these variables, their influence on problem solving behaviour and their relationship to the semantics and rules proposed here.

The intriguing differences between groups of subjects also invites further study. A replication of this study with a different sample population can shed more light on this phenomenon. A study with practicing systems and business analysts will be the most informative.

This thesis has examined UML and object-oriented design languages from the perspective of their suitability for conceptual modelling. However, this must be put into the broader context of model and modelling quality. Frameworks by (Krogstie *et al.*, 1995; Lindland *et al.*, 1994), (Moody and Shanks, 1994, 1998; Moody, 1998) and (Becker and Schütte, 1995, 1996; Rosemann and Schütte, 1997; Schütte, 1998; Schütte and Rotthowe, 1998; Rosemann, 1995; Schütte, 1999) provide some further criteria for model and modelling quality. The adequacy of the language for the task (Schütte, 1999) is but one of them. Likely the most important other criterion is that of cost and effort of modelling. Cost and effort of modelling arises due to learning and applying the modelling language and modelling rules, the construction of the model and the interpretation and comprehension of the

model. While the experimental study did not show any significant differences in the time to interpret a model, other costs may stem from learning the rules or applying the rules during model construction. These costs must be estimated by further research in order to enable a cost/benefit assessment of the proposed rules and guidelines.

Future research may also want to examine the effect of the modelling rules on other quality dimensions such as pragmatic quality with respect to the interpreter, social quality with respect to the group of interpreters, flexibility, simplicity and completeness. Implementability is a quality criterion suggested by Moody and Shanks (1998) and partially addressed by the results of the case study. The effects on comparability, clarity and relevance as proposed by Becker and Schütte (1996) also remain to be empirically assessed.

Chapter 14

Conclusion

This thesis set out to provide a language for conceptual modelling. This language is intended to be usable both for conceptual and IS design, in order to alleviate potential problems when translating from one language to another. Since object-oriented languages are well accepted for IS design, this led to the question of whether and how their use can be extended to include conceptual modelling of the business and organization. The Unified Modelling Language is used as a specific example of object-oriented languages.

This thesis investigated the question from an ontological perspective, first using theoretical deductive methods. This led to real-world semantics and ontological rules for conceptual modelling. In a second step, empirical methods were applied to corroborate the applicability and usefulness of the theoretical results. We have shown that the rules and semantics are applicable as well as beneficial to real-world modelling. They are generalizable to other object-oriented languages and enhance the domain understanding that is imparted by conceptual models and remain valid starting points for IS design.

The important differences between the BWW-ontology and object-oriented languages lie in the areas of message passing, which is a concept related to software design, and in the area of encapsulation, which is incompatible with the ontological concept of mutual properties. Both of these language areas have been investigated and interpretation and representation mappings have been found to provide guidelines to the modeller even in these areas.

In conclusion, based on the results of this research, the answer to the research question posed in Chapter 1 is a resounding Yes, object-oriented

languages are usable for conceptual modelling and this research provides guidelines on how to use them. This research is an encouraging step in the process of bridging the gulf between conceptual and design modelling. It provides a well-known language for the conceptual modeller and helps bridge the gulf between the business analyst and the software designer.

Bibliography

Agarwal, R., Sinha, A. P., and Tanniru, M. (1996). Cognitive fit in requirements modeling: A study of object and process methodologies. *Journal of Management Information Systems*, **13**(2), 137–162.

Agarwal, R., De, P., and Sinha, A. P. (1999). Comprehending object and process models: An empirical study. *IEEE Transactions of Software Engineering*, **25**(4), 541–556.

Anderson, J. and Pirolli, P. (1984). Spread of activation. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, **10**, 791–798.

Anderson, J. R. (1995). *Cognitive Psychology and its implications*. W.H. Freeman and Company, New York, NY., fourth edition.

Angeles, P. (1981). *Dictionary of Philosophy*. Harper Perennial, New York, NY.

Arango, G. (1989). Domain analysis: From art form to engineering discipline. In *Proceedings of the fifth international workshop of software specification and desing*, pages 152–159.

Auramaki, E., Lehtinen, E., and Lyytinen, K. (1988). A speech-act-based office modeling approach. *ACM Transactions on Office Information Systems*, **6**(2), 126–152.

Austin, J. (1962). *How to do things with words*. Harvard University Press, Cambridge, MA.

Barker, H., Grant, P., Jobling, C., and Townsend, P. (1993). The object-oriented paradigm: a means for revolutionising software development. *Computing and Control Engineering Journal*, pages 10–14.

Baron, J. and Li, Y. (2001). *Notes on the use of R for psychology experiments and questionnaires.*

Batini, C. and Lenzerini, M. (1986). A comparativve analysis of methodologies for database schema integration. *ACM Computer Surveys*, **18**(4).

Batra, D., Hoffer, J. A., and Bostrom, R. P. (1990). Comparing representations with relational and EER models. *Communications of the ACM*, **33**(2), 126–139.

Becker, J. and Schütte, R. (1995). Grundsätze ordnungsmäßiger Modellierung. *Wirtschaftsinformatik*, **5**, 435–445.

Becker, J. and Schütte, R. (1996). *Handelsinformationssysteme.* Verlag Moderne Industrie, Landsber/Germany.

Benbasat, I. (1989). Laboratory experiments in informatin systems studies with a focus on individuals: A critical appraisal. In I. Benbasat, editor, *The Information Systems Research Challenge: Experimental Research Methods.* Harvard Business School, Boston, MA.

Bezivin, J. and Muller, P. (1999). UML: The birth and rise of a standard modeling notation. In *The Unified Modeling Language UML'98: Beyond the notation, First International Workshop, Mulhouse, France, June 1998.*

Bodart, F. and Weber, R. (1996). Optional properties versus subtyping in conceptual modeling: A theory and empirical test. In *Proceedings of the International Conference on Information Systems, Dec. 16-18, 1996*, page 450.

Bodart, F., Sim, M., Patel, A., and Weber, R. (2001). Should optional properties be used in conceptual modelling? A theory and three empirical tests. *Information Systems Research*, **12**(4).

Booch, G. (1994). *Object Oriented Analysis and Design with Applications.* Benjamin/Cummings, Redwood City, CA.

Breu, R., Hinkel, U., Hofmann, C., Klein, C., Paech, B., Rumpe, B., and Thurner, V. (1997). Towards a formalization of the unfied modeling language. In M. Aksit and S. Matsuoka, editors, *Proceedings of ECOOP'97 - 11th European Conference on Object-Oriented Programming, Jyväskylä, Finland, June 1997.* Springer Verlag.

Breu, R., Grosu, R., Hofmann, C., Huber, F., Krüger, I., Rumpe, B., Schmidt, M., and Schwerin, W. (1998a). Exemplary and complete object interaction descriptions. *Computer Standards and Interfaces*, **19**, 335–345.

Breu, R., Grosu, R., Huber, F., Rumpe, B., and Schwerin, W. (1998b). Systems, views and models of UML. In M. Schader and A. Korthaus, editors, *The Unified Modeling Language, Technical Aspects and Applications*. Physica Verlag, Heiderberg.

Bruel, J.-M. and France, R. B. (1999). Transforming UML models to formal specifications. In *The Unified Modeling Language UML'98: Beyond the notation, First International Workshop, Mulhouse, France, June 1998*.

Bunge, M. A. (1977). *Ontology I: The Furniture of the World*, volume 3 of *Treatise On Basic Philosophy*. D. Reidel Publishing Company, Dordrecht, Holland.

Bunge, M. A. (1979). *Ontology II: A World of Systems*, volume 4 of *Treatise On Basic Philosophy*. D. Reidel Publishing Company, Dordrecht, Holland.

Capretz, L. F. (2003). A brief history of the object-oriented approach. *Software Engineering Notes*, **28**(2).

Chisholm, R. (1996). *A Realistic Theory of Categories - An Essay on Ontology*. Cambridge University Press, Cambridge.

Churchill, G. (1979). A paradigm for developing better measures of marketing constructs. *Journal of Marketing Research*, **16**, 64–73.

Coad, P. and Yourdon, E. (1990). *Object-Oriented Analysis*. Yourdon Press, Englewood Cliffs, NJ.

Cockroft, S. and Rowles, S. (2003). Ontological evaluation of health models: Some early findings. In *Proceedings of the 7th Pacific Asia Conference on Information Systems, 10-13 July, Adelaide, Australia*.

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, **20**(1), 37–46.

Cohen, S. and Northrop, L. (1998). Object-oriented technology and domain analysis. In P. Devanbu and J. Poulin, editors, *Proceedings of the Fifth International Conference on Software Reuse*, pages 86–93.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., and Gilchrist, H. (1994). *Object-Oriented Development: The Fusion Method*. Prentice-Hall, Engelwood Cliffs, NJ.

Collins, A. and Loftus, E. (1975). A spreading activation theory of semantic processing. *Psychological Review*, **82**, 407–428.

Collins, A. and Quillian, M. (1969). Retrieval time from semantic memory. *Journal of Verbal Learning and Verbal Behavior*, **8**, 240–248.

Collins, A. and Quillian, M. (1970). Facilitating retrieval from semantic memory: The effect of repeating part of an inference. *Acta Psychologica*, **33**, 304–314.

Davies, I., Green, P., Milton, S., and Rosemann, M. (2003). Using meta models for the comparison of ontologies. In *Proceedings of the 15th Conference on Advanced Information Systems Engineering, CAiSE 2003, Velden, Austria*.

Davis (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly*, **13**, 319–340.

Dussart, A., Aubert, B. A., and Patry, M. (2002). An evaluation of inter-organizational workflow modeling formalisms. Working paper, Ecole des Haute Etudes Commercials Montreal.

Evans, A. and Clark, A. (1998). Foundations of the unified modeling language. In *2nd Northern Formal Methods Workhop, Ilkley, Electronic Workshops in Computing*. Springer Verlag.

Evans, A. and Kent, S. (1999). Core meta-modelling semantics of UML: the pUML approach. In *UML'99 The Unified Modeling Language - Beyond the Standard: Second International Workshop, Fort Collins, CO, October 28-30, 1999*, pages 99–115.

Evans, A., Lano, K., France, R., and Rumpe, B. (1999a). Meta-modelling semantics of UML. In H. Kilov, editor, *Behavioural Specifications for Businesses and Systems*. Kluwer.

Evans, A., France, R., Lano, K., and Rumpe, B. (1999b). The UML as a formal modeling notation. In *The Unified Modeling Language UML'98: Beyond the notation, First International Workshop, Mulhouse, France, June 1998*.

Evans, A. S. (1998). Reasoning with UML class diagrams. In *Workshop on Industrial Strength Formal Methods. WIFT'98 Florida*. IEEE Press.

Evermann, J. and Wand, Y. (2001a). An ontological examination of object interaction in conceptual modeling. In *Proceedings of the Workshop on Information Technologies and Systems WITS'01, New Orleans, December 15-16, 2001.*

Evermann, J. and Wand, Y. (2001b). Towards ontologically based semantics for UML constructs. In H. Kunii, S. Jajodia, and A. Solvberg, editors, *Proceedings of the 20th International Conference on Conceptual Modeling, Yokohama, Japan, Nov. 27-30, 2001, ER 2001.*

Fettke, P. and Loos, P. (2003). Ontological evaluation of reference models using the Bunge-Wand-Weber model. In *Proceedings of the 2003 Americas Conference on Information Systems, August 4-6, Tampa, FL.*

Fil, W. G. (1999). Wither object orientation? What is object orientation, anyway? *ACM SIGAPL APL Quote Quad*, **30**(2), 3–6.

Fowler, M. and Kendall, S. (2000). *UML Distilled : A Brief guide to the standard object oriented modelling language*. Addison Wesley, Reading, MA.

Galfione, P., Galdiolo, A., Valreio, A., and Cardino, G. (2000). Exploiting enterprise knowledge through domain analysis and frameworks: An experimental work. In A. Tjoa, R. Wagner, and A. Al-Zobaidie, editors, *Proceedings of the 11th international Workshop on Database and Expert Systems Applications, 2000*, pages 813–817.

Gemino, A. (1999). *Empirical Comparisons of Systems Analysis Modeling Techniques*. Ph.D. thesis, University of British Columbia, Canada.

Gemino, A. and Wand, Y. (2001). Comparing object oriented with structured analysis techniques in conceptual modeling. Draft Manuscript.

Gomaa, H. (1992). An object-oriented domain analysis and modeling method for software reuse. In V. Milutinovic, B. Shriver, J. Nunamaker, and R. Sprague, editors, *Proceedings of the Twenty-Fifth Hawaii International Conference of System Sciences, 1992*, pages 46–52, Vol. 2.

Gopnik, A. (2001). Theories, language and culture. In M. Bowerman and S. Levinson, editors, *Language Acquisition and Conceptual Development*, pages 45–69. Cambridge University Press, New York, NY.

Green, P. and Rosemann, M. (2000). Ontological analysis of integrated process modelling. *Information Systems*, **25**(2).

Gruninger, M. and Lee, J. (2002). Ontology applications and design. *Communications of the ACM*, **45**(2), 39–41.

Guarino, N. and Welty, C. (2002). Evaluating ontological decisions with ontoclean. *Communications of the ACM*, **45**(2), 61–65.

Harel, D. (1988). On visual formalisms. *Communications of the ACM*, **31**(5), 514–530.

Harel, D. and Gery, E. (1996). Executable object modeling with statecharts. In *Proceedings of ICSE-18*, pages 246–256.

Harel, D. and Kupferman, O. (2000). On the behavioral inheritance of state-based objects. In *Proceedings of the 34th international conference on Technology of Object-Oriented Languages, 2000. TOOLS 34*, pages 83–94.

Harman, H. H. (1976). *Modern Factor Analysis*. The University of Chicago Press, Chicago, IL, 3rd edition.

Hirsh-Pasek, K., Reeves, L. M., and Golinkoff, R. (1993). Words and meaning: From primitives to complex organization. In J. Gleason and N. Ratner, editors, *Psycholinguistics*, pages 133–197. Holt, Rinehart, &Winston, Orlando, FL.

Hoffer, J. A., George, J. F., and Valacich, J. S. (2002). *Modern Systems Analysis and Design*. Prentice Hall, Upper Saddle River, NJ.

Holsaple, C. W. and Joshi, K. (2002). A collaborative approach to ontology design. *Communications of the ACM*, **45**(2), 42–47.

Howell, D. (1997). *Statistical Methods for Psychology*. Duxbury Press, Delmont, CA.

Hufnagel, E. M. and Conca, C. (1994). User response data: The potential for errors and biases. *Information Systems Research*, **5**(1), 48–73.

Ihaka, R. and Gentleman, R. (1996). R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, **5**(3), 299–314.

Jacobson, I. (1992). *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley, Wokingham.

Jayase, T., Ikeda, N., and Matsumoto, K. (2001). A three-view model for developing object-oriented frameworks. In Q. Li, R. Riehle, G. Pour, and B. Meyer, editors, *39th international conference and exhibition on technology of object-oriented languages and systems, 2001. TOOLS 39*, pages 108–119.

Johnson, R. A. (2002). Object-oriented systems development: A review of empirical research. *Communications of the Association for Information Systems*, **8**, 65–81.

Kim, J. and Lerch, F. J. (1997). Why is programming (sometimes) so difficult? Programming as scientific discovery in multiple problem spaces. *Information Systems Research*, **8**(1), 25–50.

Kim, J., Hahn, J., and Hahn, H. (2000). How do we understand a system with (so) many diagrams? Cognitive integration processes in diagrammatic reasoning. *Information Systems Research*, **11**(3), 284–303.

Kirstensen, B. B. and Osterbye, K. (1996). A conceptual perspective on the comparison of object-oriented programming languages. *ACM SIGPLAN Notices*, **31**(2), 42–54.

Knapp, A. (1999). A formal semantics for UML interactions. In *UML'99 The Unified Modeling Language - Beyond the Standard: Second International Workshop, Fort Collins, CO, October 28-30, 1999*, pages 116–130.

Korson, T. and McGregor, J. D. (1990). Understanding object-oriented: A unifying paradigm. *Communications of the ACM*, **33**(9), 40–60.

Krogstie, J., Lindland, O. I., and Sindre, G. (1995). Towards a deeper understanding of quality in requirements engineering. In *Proceedings of the CAISE'95 Conference*, pages 82–95.

Kuhn, T. (1996). *The structure of scientific revolutions*. The University of Chicago Press, Chicago, third edition.

Kung, C. and Solvberg, A. (1986). Activity modelling and behaviour modelling. In T. Olle, H. Sol, and A. Verrijn-Stuart, editors, *Information System Design Methodologies: Improving the Practrice*. Addison-Wesley Publishing Co., Amsterdam.

Lakoff, G. (1987). *Women, Fire, and Dangerious Things – What Categories reveal about the Mind*. The University of Chicago Press, Chicagor, IL.

Lakos, C. and Lewis, G. (2000). Behaviour inheritance for object lifecycles. In *Proceedings of the 33rd international conference on Technology of Object-Oriented Languages, 2000. TOOLS 33*, pages 262–273.

Lano, K. and Bicarregui, J. (1999). Semantics and transformations for UML models. In *The Unified Modeling Language UML'98: Beyond the notation, First International Workshop, Mulhouse, France, June 1998*, pages 107–119.

Lano, K. and Evans, A. (1999). Rigorous development in UML. In *ETAPS'99, FASE workshop*, Lecture Notes on Computer Science. Springer Verlag.

Larkin, J. H. and Simon, H. A. (1987). Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, **11**, 65–99.

LeGrand, A. (1998). Specialization of object lifecycles. In C. Rolland and G. Grosz, editors, *Proceedings of the Internationl Conference on Object Oriented Information Systems, 9-11 September 1998, Paris.*, pages 259–275.

Lilius, J. and Paltor, I. (1999). Formalising UML state machines for model checking. In *UML'99 The Unified Modeling Language - Beyond the Standard: Second International Workshop, Fort Collins, CO, October 28-30, 1999*, pages 430–445.

Lim, K. H. and Benbasat, I. (2000). The effect of multimedia on perceived equivocality and perceived usefulness of information systems. *MIS Quarterly*, **24**(3), 449–471.

Lindland, O. I., Sindre, G., and Solvberg, A. (1994). Understanding quality in conceptual modeling. *IEEE Software*, **11**(2), 42–49.

Mackworth, A. K. (1977). Consistency in networks of relations. *Artificial Intelligence*, pages 99–118.

Mayer, R. E. (1989). Models for understanding. *Review of Educational Research*, **59**(1), 43–64.

Meyer, B. (1996). The many faces of inheritance: A taxonomy of taxonomy. *IEEE Computer*, **29**(5), 105–108.

Miller, L. (2002). *Instructor's Resource Manual for Modern Systems Analysis and Design*. Pearson Education, Upper Saddle River, NJ.

Milton, S. and Kazmierczak, E. (1999). Enriching the ontological foundations of modelling in information systems. In *Proceedings of the Information Systems Foundation Workshop. Ontology, Semiotics and Practice, 1999*.

Moody, D. (1998). Metrics for evaluating the quality of entity relationship models. In *Proceedings of the ER'98 Conference*, New York, NY. Springer Verlag.

Moody, D. and Shanks, G. (1994). What makes a good data model? evalutating the quality of entity relationship models. In *Proceedings of the ER'94 Conference*, New York, NY. Springer Verlag.

Moody, D. and Shanks, G. (1998). Improving the quality of entity relationship models - experience in research and practice. In *Proceedings of the ER'98 Conference*, New York, NY. Springer Verlag.

Moore, G. C. and Benbasat, I. (1991). Development of an instrument to measure the perceptions of adopting an information technology innovation. *Information Systems Research*, **2**(3), 192–222.

Morandin, E., Stellucci, G., and Baruchelli, F. (1998). A reuse-based software process based on domain analysis and OO framework. In *Proceedings of the 24th Euromicro Conference*, pages 890–897, Vol. 2.

Morisio, M., Travassos, G., and Stark, M. (2000). Extending UML to support domain analysis. In *Proceedings of the Fifteenth IEEE International Conference on Automated Software Engineering, ASE*, pages 321–324.

Mylopoulos, J. (1992). Conceptual modeling and telos. In P. Locuopoulos and R. Zicari, editors, *Conceptual Modeling, Databases and Cases*. John Wiley & Sons, Inc, New York et. al.

Newell, A. and Simon, H. A. (1972). *Human Problem Solving*. Prentice Hall, Englewood Cliffs, NJ.

Norman, D. (1986). Cognitive engineering. In D. Norman and S. Draper, editors, *User Centered Design: New Perspectives on Human Computer Interaction*. Lawrence Erlbaum Associates, Hillsdale, NJ.

Noy, N. F. and Hafner, C. D. (1997). The state of the art in ontology design: A survey and comparatie review. *AI Magazine*, **18**(3), 53–74.

Nyerges, T., Moore, T., Montejano, R., and Compton, M. (1998). Developing and using interaction coding systems for studying groupware use. *Human-Computer Interaction*, **13**, 127–165.

OMG (2001). *The Unified Modelling Language Specification. Version 1.4*. OMG.

Opdahl, A. and Henderson-Sellers, B. (1999). Evaluating and improving OO modelling languages using the BWW-model. In *Proceedings of the Information Systems Foundation Workshop 1999*. www.comp.mq.edu.au/isf99/Opdahl.htm.

Opdahl, A. and Henderson-Sellers, B. (2002). Ontological evaluation of the UML using the Bunge-Wand-Weber model. *Software and Systems Modeling*, **1**(1), 43–67.

Opdahl, A. and Sindre, G. (1993). Concepts for real-world modelling. In *Proceedings of Conference on Advanced Information Systems Engineering CAiSE'93*, pages 309–327.

Opdahl, A., Henderson-Sellers, B., and Barbier, F. (1999). An ontological evaluation of the OML metamodel. In E. Falkenberg and K. Lyytinen, editors, *Information System Concepts: An Integrated Discipline Emerging*. IFIP/Kluwer.

Opdahl, A. L. and Henderson-Sellers, B. (2001). Grounding the OML metamodel in ontology. *The Journal of Systems and Software*, **57**(2), 119–143.

Övergaard, G. (1999). A formal approach to collaborations in the unified modeling language. In *UML'99 The Unified Modeling Language - Beyond the Standard: Second International Workshop, Fort Collins, CO, October 28-30, 1999*, pages 99–115.

Övergaard, G. and Palmkvist, K. (1999). A formal approach to use cases and their relationships. In *The Unified Modeling Language UML'98: Beyond the notation, First International Workshop, Mulhouse, France, June 1998*, pages 406–418.

Parsons, J. and Wand, Y. (1991). The object paradigm – two for the price of one? In *Proceedings of the Workshop on Information Technology and Systems WITS 1991*, pages 308–319.

Parsons, J. and Wand, Y. (1997). Using objects for systems analysis. *Communications of the ACM*, **40**(12), 104–110.

Philippow, I. and Riebisch, M. (2001). Systematic definition of reusable architectures. In *Proceedings of the Eigth Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, ECBS*, pages 128–135.

Pinheiro, J. and Bates, D. (2000). *Mixed-Effects Models in S and S-PLUS*. Springer Verlag, New York, NY.

Quine, W. v. O. (1953). Two dogmas of empiricism. In *From a Logical Point of View*. Harvard University Press, Cambridge, MA.

Rahm, E. and Bernstein, P. A. (2001). A survey of approaches to automatic schema matching. *The VLDB Journal - The International Journal on Very Large Databases*, **10**(4), 334–350.

Rosemann, M. (1995). Beschreibung und Gestaltung der Produktion auf der Basis der Grundsätze Ordnungsmäßiger Modellierung. Working paper, University of Muenster, Germany.

Rosemann, M. and Green, P. (1999). Enhancing the process of ontological analysis – the who cares dimension. In *Proceedings of the Information Systems Foundation Workshop. Ontology, Semiotics and Practice, 1999*.

Rosemann, M. and Green, P. (2000). Developing a meta-model for the Bunge-Wand-Weber (BWW) ontological constructs. Technical report, Department of Commerce, University of Queensland.

Rosemann, M. and Green, P. (2002). Developing a meta-model for the Bunge-Wand-Weber ontological constructs. *Information Systems*, **27**(2), 75–91.

Rosemann, M. and Schütte, R. (1997). Grundsätze ordnungsmäßiger Referenzmodellierung. Working paper, University of Muenster, Germany.

Rosemann, M. and zur Mühlen, M. (1998). Evaluation of workflow management systems – a meta-model approach. *Australian Journal of Information Systems*, **6**(1), 103–116.

Rumbaugh, J. *et al.* (1991). *Object Oriented Modeling and Design*. Prentice Hall, Englewood Cliffs, NJ.

Scheer, A.-W. (1999). *ARIS – Business Process Modeling*. Springer Verlag, Berlin.

Scheffe, H. (1959). *Analysis of Variance*. Wiley, New York, NY.

Schrefl, M. and Stumptner, M. (2002). Behavior-consistent specialization of object life cycles. *ACM Transactions of Software Engieneering and Methodology*, **11**(1), 92–148.

Schütte, R. (1998). *Grundsätze ordnungsmäßiger Referenzmodellierung*. Gabler Verlag, Wiesbaden, Germany. Also PhD Dissertation, University of Münster, Germany.

Schütte, R. (1999). Architectures for evaluating the quality of information models - a meta and an object level comparison. In *Proceedings of the ER'99 Conference*, New York, NY. Springer Verlag.

Schütte, R. and Rotthowe, T. (1998). The guidelines of modeling - an approach to enhance the quality in information models. In *Proceedings of the ER'98 Conference*, pages 240–254.

Searle, J. (1969). *Speech Acts*. Cambridge University Press, Cambridge, UK.

Shavelson, Richard, J. (1996). *Statistical Reasoning for the Behavioral Sciences*. Allyn and Bacon, Needham Heights, MA, 3rd edition.

Sinha, A. P. and Vessey, I. (1992). Cognitive fit: An emprirical study of recursion and iteration. *IEEE Transactions of Software Engineering*, **18**(5), 368–379.

Smith, B. and Welty, C. (2001). Ontology: Towards a new synthesis. In *Proceedings of the Second International Conference on Formal Ontology and Information Systems, FOIS'01, October 17-19, Qgunquit, Maine*, pages iii–ix.

Straub, D. W. (1989). Validating instruments in MIS research. *MIS Quarterly*, **13**(2), 147–169.

Tailvalsaari, A. (1996). On the notion of inheritance. *ACM Computing Surveys*, **28**(3), 438–479.

Test Manager (2001). *Prentice Hall Test manager for Hoffer, George Valacich - Modern Systems Analsysis and Design, 3rd edition*. Prentice Hall.

Tukey, J. (1977). *Exploratory Data Analysis*. Addison-Wesley, Reading, MA.

Uschold, M. and Gruninger, M. (1996). Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, **11**(2).

Van Hentenryck, P. (1989). *Consistency techniques in logic programming*. MIT Press, Cambridge, MA.

Vessey, I. (1991). Cognitive fit: A theory-based analysis of the graphs versus tables literature. *Decision Sciences*, **22**(2), 219–240.

Vessey, I. and Conger, S. A. (1994). Requirements specification: Learning object, process and data methodologies. *Communications of the ACM*, **37**(5), 102–113.

Wand, Y. (1989). A proposal for a formal model of objects. In W. Kim and F. Lchovsky, editors, *Object-oriented concepts, languages, applications and databases*, pages 537–559. ACM Press. Addison-Wesley.

Wand, Y. and Wang, R. (1995). Anchoring data quality dimensions in ontological foundations. *Communications of the ACM*, **39**(11), 86–95.

Wand, Y. and Weber, R. (1989). An ontological evaluation of systems analysis and design methods. In E. Falkenberg and P. Lingreen, editors, *Information System Concepts: An In-Depth Analysis*. Elsevier Science Publishers B.V., North-Holland.

Wand, Y. and Weber, R. (1990). Mario Bunge's ontology as a formal foundation for information systems concepts. In P. Weingartner and G. Dorn, editors, *Studies on Mario Bunge's Treatise*. Rodopi, Atlanta.

Wand, Y. and Weber, R. (1993). On the ontological expressiveness of information systems analysis and design grammars. *Journal of Information Systems*, (3), 217–237.

Wand, Y. and Woo, C. (1999). Ontology-based rules for object-oriented enterprise modelling. Working paper 99-MIS-001, Faculty of Commerce and Business Administration, University of British Columbia.

Wand, Y., Storey, V. C., and Weber, R. (1999). An ontological analysis of the relationship construct in conceptual modeling. *ACM Transactions on Database Systems*, **24**(4), 494–528.

Wand, Y., Woo, C., and Jung, D. (2000). Object-oriented modeling: From enterprise model to logical design. In *Proceedings of the Tenth Annual Workshop on Information Technologies and Systems (WITS'00)*, Brisbane.

Wang, S. (1996). Two MIS analysis methods: An experimental comparison. *Journal of Education for Business*, **71**(3), 136–141.

Weber, R. and Zhang, Y. (1996). An analytical evaluation of NIAM's grammar for conceptual schema diagrams. *Information Systems Journal*, **6**(2), 147–170.

Wegner, P. (2003). Dimensions of object-based language design. In *International Conference on Object-Oriented Programming Systems, Languages and Applications, OOPSLA 87*, pages 168–182.

Yadav, S. B., Bravoco, R. R., Chatfield, A. T., and Rajkumar, T. M. (1988). Comparison of analysis techniques for information requirement determination. *Communications of the ACM*, **31**(9), 1090–1097.

Appendix A

The Object Constraint Language

The object constraint language is a semi-formal language that allows the modeller to express constraints on the use of other language elements. It provides ways of referencing elements of a UML model and is widely used in the specification of the UML meta-model itself.

Any OCL expression must be associated with a context. This context is usually a model element which can then be referred to with the keyword `self`. Within this context, an OCL expression can be used to specify invariants, expressed with the keyword `inv`, functions and pre- and post-conditions[1]. An invariant is a condition that must always hold for a given context. An OCL function returns a result of a specified result type. The result can either be assigned in Pascal like manner by assigning it to a variable with the name of the function, e.g.

(A.1)

```
context Foo:
Bar(Foo1 : Foo, Foo2 : Foo) : Boolean
 ...
 Bar = ....
```

This expression declares a function `Bar` for the type `Foo` which returns

[1] Pre- and post-conditions can be used for the specification of methods or operations.

a boolean value. `Foo` is an element declared in a model, such as a class or attribute, etc. The function is declared to take two parameters, `Foo1` and `Foo2` both of type `Foo`. Another way of expressing this is by using the keyword `result`:

(A.2)

```
context Foo:
Bar(Foo1 : Foo, Foo2 : Foo) : Boolean
 ...
 result = ....
```

OCL provides constructs for conditional evaluation of statements: `'if ... then ... else ... endif'`. One key feature of OCL is the built-in type `collection` and associated operations on collections. Functions of collections are referred to using the operator `->`, e.g.

(A.3)

```
context Foo inv:
   Foo->size() > 1
```

The following are the main functions defined for any collection in OCL:

```
collection1->exists(f : Foo | ...logical expression...)
```

returns `true` if collection1 contains an element `f` of type `Foo` that satisfies the logical expression specified. Collection1 must be a collection of elements of type `Foo`.

```
collection1->select(f : Foo | ...logical expression...)
```

returns the sub-collection of elements `f` of type `Foo` that satisfy the logical expression and are elements of collection1.

```
collection1->reject(f : Foo | ...logical expression...)
```

returns the sub-collection of elements `f` of type `Foo` that *do not* satisfy the logical expression and are elements of collection1.

```
collection1->forall(f : Foo | ...logical expression...)
```

returns `true` if the logical expression holds for all elements `f` of type `Foo` that are elements of collection1.

```
collection1->includes(f : Foo | ...logical expression...)
```

returns `true` if collection1 includes all the elements `f` of type `Foo` for which the logical expression holds.

```
collection1->excludes(f : Foo | ...logical expression...)
```

returns `true` if collection1 excludes all the elements `f` of type `Foo` for which the logical expression holds.

```
collection1->union(collection2)
```

returns the union of elements of collection1 and collection2 where the elements of the two collections must be of the same type.

```
collection1->size()
```

returns the number of elements of collection1.

```
collection1->isEmpty()
```

returns `true` if collection1 does not contain any elements.

Functions can be combined as in the following example:

(A.4)

context `Foo` **inv**:
```
self->select(b : Bar | ...expression1...)->size() > 0
```

This specifies an invariant that ensures that there is at least on element `b` of the collection `Foo` of elements of type `Bar` that satisfies expression1.

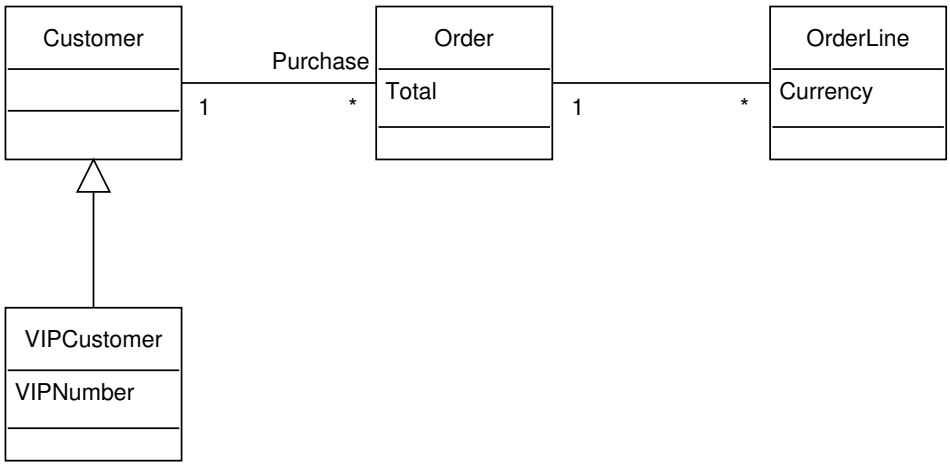


Figure A.1: Example class diagram

OCL expressions can refer to UML models and associations between model elements can be dereferenced by using the `'.'` operator as in the following example that refers to Fig. A.1.

(A.5)

context `Customer` **inv**:
`self.purchase.orderline.currency="US$"`

Note that for associations with rolenames the opposite element is referenced through the role name (e.g. 'purchase') whereas for associations without rolenames, the opposite element is referenced by its name (e.g. 'orderLine'). Note also that the references always begin with a lowercase letter.

Consecutive references to collection typed attributes are 'collapsed'. In the above example `self.purchase.orderline` is the set of all orderlines for all purchases by the specific customer (OMG, 2001, 6.6.2.1).

An OCL expression of some element can refer to any feature of the supertype by using the keyword `parent` as in the following example, again referring to Fig. A.1:

(A.6)

```
context VIPCustomer inv:
self.parent.purchase.total > 1000
```

Since OCL is a typed language, it provides the following functions for determining and recasting the type of any referenced element.

- `Bar.oclIsTypeOf(Foo)` returns `true` if `Bar` is of type `Foo` or a subtype thereof.
- `Bar.oclIsKindOf(Foo)` returns `true` if `Bar` is of type `Foo` but will return `false` if `Bar` is of a subtype of `Foo`.
- `Bar.oclAsKind(Foo)` recasts the type of `Bar` as type `Foo`.

As an example consider again the class diagram in Fig. A.1. We can then specify

(A.7)

```
context Customer:
VIPNumber() : Integer
if self.oclIsKindOf(VIPCustomer) then
   result=self.oclAsKind(VIPCustomer).VIPNumber
else
   result=0
endif
```

Appendix B

Auxilliary OCL Functions

This appendix defines some useful OCL expressions that are used in various places in the discussion.

Expression (B.1) defines all those attributes as immediate properties which are defined with a class and with association classes which the class participates in.

(B.1)

```
context Class :
let AllImmediateProperties : Set(Attribute);
AllImmediateProperties =
  self.allAttributes()
  ->union(
    self.allAssociations
    ->select(as | as.oclIsTypeOf(AssociationClass))
    .oclAsType(Class).allAttributes())
```

Expression (B.2) defines all those attributes as PartProperties which are, recursively, defined for all parts of a class that participates as aggregate in an aggregation relationship.

(B.2)

```
context Class :
let PartProperties : Set(Attribute);
if self.association
   ->select(as | as.aggregation="aggregate"
   .association.connection
   ->reject(conn | conn=self.association)
   ->exists()
then
PartProperties =
   self.AllImmediateProperties
   ->union(
     self.assocation
    ->select(as | as.aggregation="aggregate")
    .asssocation.connection
    ->reject(conn | conn=self.association)
    .participant.allPartProperties)
else
PartProperties =
   self.AllImmediateProperties
```

Expression (B.3) defines all properties of a class to be the union of the properties of itself, assocation classes it participates in, and the properties of direct and indirect parents of generalizations, but *not* the properties of its own parts.

(B.3)

```
context Class :
let AllNonPartProperties : Set(Attribute);
if self.parent->exists() then
AllInheritedProperties =
   self.allImmediateProperties->union(
   self.parent.allInheritedProperties)
else
AllNonPartProperties =
   self.allImmediateProperties
```

Expression (B.4) defines all properties of a class to be the union of the properties of itself, assocation classes it participated in, all direct and indirect parts and the properties of direct and indirect parents of generalizations. The latter in turn includes properties of parts, association classes etc.

(B.4)

```
context Class :
let AllInheritedProperties : Set(Attribute);
if self.parent->exists() then
AllInheritedProperties =
   self.PartProperties->union(
   self.parent.allInheritedProperties)
else
AllInheritedProperties =
   self.PartProperties

context Class :
let AllProperties : Set(Attribute);
AllProperties =
  self.allInheritedProperties
```

OCL expression (B.5) returns the set of all direct or indirect parts of an aggregate, including the class itself.

(B.5)

```
context Class :
let AllParts : Set(Class);
if self.association
   ->select(as | as.aggregation="aggregate"
   .association.connection
   ->reject(conn | conn=self.association)
   ->exists()
then
AllParts = self
   ->union(
     self.assocation
    ->select(as | as.aggregation="aggregate")
    .asssocation.connection
    ->reject(conn | conn=self.association)
    .participant)
else
AllParts = self
```

The followin OCL expression defines all direct and indirect generalizations that a class inherits from:

(B.6)

```
context Class:
Let AllParents: set(Class);
self.parent->union(self.parent->allParents())
```

The following OCL expression defines all direct or indirect aggregates that a certain class is part of, including aggregations that are inherited from parent classes.

(B.7)

```
context Class:
Let PartOf: set(Class);
self.association
  ->select(as | as.aggregation="aggregate")
  .association.connection
  ->reject(conn | conn=self.association)
->union(self.allParents())
->union(
self.association
  ->select(as | as.aggregation="aggregate")
  .association.connection
  ->reject(conn | conn=self.association)
  ->partOf()
->union(self.allParents()->allParts())
```

The following OCL expressions (B.8) and (B.9) define an operation on states and state transitions which yields the associated top-most state machine, either directly or recursively up a substate hierarchy.

(B.8)

```
context State
let Machine() : StateMachine
result =
  if
      self.container->notEmpty()
  then
      self.container.Machine()
  else
      self.StateMachine
  endif
```

(B.9)

```
context Transition
let Machine() : StateMachine
result = self.state.Machine()
```

Expression (B.10) yields the class that owns the transition, either as a top-level transition or as a transition of some class description.

(B.10)

```
context Transition
Let Transition::Owner() : Class
If
  transition.Machine().context.oclIsTypeOf(Class)
then
  result=transition.Machine().context
else
  result=transition.Machine().context.owner
endif
```

Expression (B.11) defines a transition to be a transition of a top-level state chart if it is owned, directly or indirectly, by a state machine that specifies the behaviour of a class.

(B.11)

```
context Transition:
Let Transition::IsTopLevel() : boolean
If
  self.Machine().context.oclIsTypeOf(Class)
then
  result=true
else
  result=false
endif
```

Expression (B.12) defines a state to be a state of a top-level state chart if it is owned, directly or indirectly, by a state machine that specifies the behaviour of a class.

(B.12)

```
context State:
Let State::IsTopLevel() : boolean
If
  self.Machine().context.oclIsTypeOf(Class)
then
  result=true
else
  result=false
endif
```

Expression (B.13) define a function that indicates whether a state machine contains (directly or indirectly) a state transition changing the state of a thing.

(B.13)

```
context StateMachine::OneTransition() : Boolean
result =
   if self.top.internalTransition
       ->exists(t : Transition |
         t.source <> t.target and
         t.source.oclIsTypeOf(State) and
         t.target.oclIsTypeOf(State) )
   or
     self.top.submachine
       ->exists(sm : StateMachine | sm->OneTransition())
   then true
   else false
   endif
```

Expression (B.14) defines a function of an attribute that yields the set of owners. If the attribute is owned by a class, this class is the owner. If the attribute is an association class attribute, the owners are the classes participating in the association.

(B.14)

```
context Attribute
Let Attribute::PropertyOf() : Set (Class)
if self.owner.oclIsTypeOf(Class)
then
   result=self.owner
else
   result=self.connection.participant
endif
```

The following expression (B.15) returns the value of some attribute in some state.

(B.15)

```
context State:
Let State::AttributeValue(a : Attribute) : value
result = self.attributevalue->select(av : av.attribute=a)
```

The following expression (B.16) defines a function on transitions that indicates whether a transition changes attributes of association classes, representing mutual properties. It checks whether the number of those attributes that do not belong to the owner of the transition (i.e. belong to an association class) and that possess a different value in the source state than in the target state, is greater zero. It makes use of expressions (B.10) and (B.15).

(B.16)

```
context Transition:
Let Transition::ChangeMutualProperties() : boolean
if
  self.target.attributeLink.attribute
  ->reject(a | a.propertyOf()=self.owner())
  ->reject(a | self.source.attributeValue(a) =
               self.target.attributeValue(a))
  ->size() > 0
then
  result = true
else
  result = false
endif
```

(B.17)

```
context Transition:
Let Transition::ChangeEmergentProperties() : boolean
if
  self.target.attributeLink.attribute
  ->reject(a | self.source.attributeValue(a) =
               self.target.attributeValue(a))
  ->reject(a.owner.association
                 ->select(ae | ae.aggregation="aggregate")
                 ->size() > 0 )
  ->size() > 0
and

then
  result = true
else
  result = false
endif
```

The following expressions (B.18) and (B.19) compute the set of inital or final states of a given state machine.

(B.18)

```
context StateMachine:
Let StateMachine::AllInitialStates() : set (State)
if self.top.oclIsTypeOf(CompositeState) and
   self.top.isConcurrent
then
   result=self.top.subvertex.subvertex->select(sv |
            sv.incoming.source.oclIsKindOf(PseudoState)
              and
            sv.incoming.source.kind="initial")
else
   result=self.top.subvertex->select(sv |
            sv.incoming.source.oclIsKindOf(PseudoState)
              and
            sv.incoming.source.kind="initial)
endif
```

(B.19)

```
Let StateMachine::AllFinalStates() : set (FinalState)
if self.top.oclIsTypeOf(CompositeState) and
   self.top.isConcurrent
then
   result=self.top.subvertex.subvertex->select(sv |
            sv.oclIsKindOf(FinalState))
else
   result=self.top.subvertex->select(sv |
            sv.oclIsKindOf(FinalState))
endif
```

Appendix C

List of Rules and Corollaries

This appendix contains a summary list of the rules and corollaries developed in chapters 4, 5 and 6. The rules are listed in the order in which they are derived in those chapters, to ensure logical continuity.

Rule 1 *Only substantial entities in the world are modelled as objects.*

Rule 2 *Ontological properties of things must be modeled as UML-attributes.*

Corollary 1 *Attributes in a UML-description of the real world cannot refer to substantial entities.*

Rule 3 *Sets of mutual properties must be represented as attributes of association classes.*

Corollary 2 *An association class cannot represent substantial entities or composites of substantial entities.*

Corollary 3 *If an association class of an n-ary association is intended to represent substantial things, the association should instead be modelled as one with arity (n+1).*

Corollary 4 *An association class representing a composite must instead be modelled as a composite with attributes representing emergent intrinsic properties.*

Corollary 5 *An association class cannot possess methods or operations.*

Corollary 6 *An association class cannot be associated with a state machine.*

Corollary 7 *An association class must possess at least one attribute.*

Corollary 8 *An association class must not be associated with another class.*

Corollary 9 *An association class must not participate in generalization relationships.*

Rule 4 *If mutual properties can change quantitatively, methods and operations that change the values of attributes of the association class must be modelled for one or more of the classes participating in the association, objects of which can effect the change, not for the associations class.*

Rule 5 *An association class represents a set of mutual properties arising out of the same interaction.*

Rule 6 *A composition relation must not be modelled.*

Rule 7 *Every UML-aggregate must possess at least one attribute which is not an attribute of its parts or participate in an association.*

Rule 8 *All UML-classes must possess at least one attribute or participate in an association.*

Rule 9 *Object ID's must not be modelled as attributes.*

Rule 10 *The set of attribute values (representing mutual and intrinsic properties) must uniquely identify an object.*

Rule 11 *Every attribute has a value.*

Corollary 10 *Attribute multiplicities greater than one imply that the order of the different individual attribute value components is semantically irrelevant.*

Rule 12 *Classes of objects that exhibit additional behaviour, additional attributes or additional association classes with respect to other objects of the same class, must be modelled as specialized sub-classes.*

Corollary 11 *An object acquiring additional behaviour or properties must be destroyed as instance of the general class and created as instance of the specialized class that is modelled with the relevant operations or association classes.*

Corollary 12 *Re-classification occurs only within a generalization / specialization hierarchy.*

Rule 13 *Every UML-aggregate object must consist of at least two parts.*

Rule 14 *An instance of a class that by virtue of additional aggregation relationships acquires emergent properties or emergent behaviour must be modelled as an instance of a specialized class which declares the corresponding attributes and operations.*

Rule 15 *Object creation occurs when an entity acquires a property so that it becomes a member of a different class.*

Corollary 13 *Object destruction occurs when an entity loses a property that is necessary for membership in a particular class.*

Rule 16 *Attributes with class scope should instead be modelled as attributes of an aggregate representing the objects of the class.*

Rule 17 *If a class that is specialized is declared as abstract, the specialization must be declared to be 'complete'.*

Rule 18 *A class that is not specialized cannot be declared abstract.*

Rule 19 *A specialized class must define more attributes, more operations or participate in more associations than the general class.*

Rule 20 *Every ordinary association must be an association class.*

Rule 21 *A UML-state represents a specific assignment of values to the attributes and attribute of association classes of the objects for which the state is defined.*

Corollary 14 *A UML-transition must change the value of at least one attribute used to define the state space.*

Rule 22 *For every level of refinement of a state C, there must be an additional set of attributes in the class description or in participating association classes that change as the object transitions among the sub-states.*

Corollary 15 *For all immediate substates of a super-state, the values assigned to attributes describing the super-state are invariant and are equal to those defining the super-state.*

Corollary 16 *Concurrent sub-states require mutually disjunct sets of additional attributes in the class description or in participating association classes.*

Rule 23 *Guard conditions on transitions from the same state to non-concurrent sub-states must be mutually disjunct.*

Rule 24 *Action states are super-states of a set of sub-states. The object transitions among these while in the action state. State charts must reflect this fact.*

Corollary 17 *States must not be associated with any actions. Sub-states corresponding to different models should be used instead.*

Corollary 18 *All states in an activity diagram must be states of the same object.*

Corollary 19 *If the partitions of an activity diagram represent different objects, they must be part of a composite which is shown in the class diagram.*

Rule 25 *The quantitative object behaviour (for each model) is* entirely *describable by top-level state chart* (SC_0)

Rule 26 *All UML-transitions in SC_0 must correspond to an operation of the object which SC_0 is associated with.*

Corollary 20 *Every object must have at least one operation.*

Corollary 21 *States in SC_0 are stable.*

Corollary 22 *All UML-transitions in SC_0 must be associated with a UML-event.*

Rule 27 *An object must exhibit additional operations expressing qualitative changes, if a super- or sub-class is defined and instances can undergo changes of class to the super- or sub-class.*

Rule 28 *Methods may be described by state charts other than top-level state charts.*

Corollary 23 *A state chart describing a method must begin and end with those states in SC_0 which the operation that the method implements is a realization of.*

Corollary 24 *State transitions out of the first state of a method realizing an operation must be associated with the same event that is associated with the transition in SC_0 which represents that operation.*

Corollary 25 *A state chart either expresses the external behaviour of an object (SC_0), a method, a signal reception or is a composite state contained in another state machine.*

Rule 29 *An operation is not directly specified by state machines. Instead, the methods that implement an operation are specified by state machines.*

Corollary 26 *A state machine that specifies the behaviour of a class or a method is not contained in other state machines.*

Corollary 27 *The method corresponding to a state chart must modify the attribute values of the object corresponding to the values defined for the initial and final state of the method.*

Rule 30 *An operation must be associated with the declaration of signal reception.*

Rule 31 *The event associated with an operation must be identical to the event associated with the signal associated with the reception.*

Corollary 28 *The state machines associated with a reception and with a method specifying the implementation of an operation which is in turn associated with that reception, must possess the same initial and final states.*

Rule 32 *Acquisition (loss) of* independent *properties leads to expansion (contraction) of the things top-level state space SC_0 by an orthogonal region.*

Corollary 29 *Every object must be capable of at least one state transition or be able to undergo change of class to a super- or sub-class.*

Rule 33 *For every class of objects between which message passing is declared, there exists an association class.*

Rule 34 *Every object must be the receiver and sender of some message.*

Rule 35 *For every attribute there exists a constraint which relates this attribute to some other attribute.*

Corollary 30 *An association class cannot be sender or receiver of a message.*

Rule 36 *A constraint relates attributes of a single class or attributes of association classes the class participates in.*

Corollary 31 *A UML-state transition associated with an action must modify an association class attribute's value.*

Corollary 32 *For every interaction between UML-objects, there must exist a corresponding UML-state transition in both interacting UML-objects.*

Corollary 33 *A state transition associated with an event must modify an association class attribute's value.*

Corollary 34 *A signal event may only be associated with a transition in a top-level state chart and the initial transition of a method implementing this.*

Corollary 35 *A call event may only be associated with a transition in a top-level state chart or the initial transition of a method implementing this.*

Corollary 36 *Synchronous communication of objects implies transition to a state which cannot be left except through a state transition associated with the return signal.*

Corollary 37 *Asynchronous communication of objects with expected response implies the existence of at least one state transition caused by the object acted upon, signifying the return interaction after the state transition signifying the original communication.*

Corollary 38 *The final state transitions of any method implementing an operation that may be invoked through a call action must cause a return action.*

Corollary 39 *For the state machine of a method to contain a state transition whose effect is a return action, there must exist a corresponding state transition in a state machine of some other object whose effect is a corresponding call action.*

Appendix D

Example 2, Alternative Interpretation

This appendix provides an alternative ontological interpretation for the second example, discussed in Section 9.2. In this interpretation, the customer orders anonymous parts. The interaction of the customer is assumed to be with an employee of the company (e.g. a salesperson) rather than with product items directly (as e.g. in a supermarket). In this case, the ordering is an interaction between the customer and the employee. Thus, it is an ontological event in both of them, corresponding to a state transition (which is not shown here). This interaction gives rise to a number of mutual properties between the customer and the employee. We can express these by rules 3 and 5 (mutual properties due to interaction as association class attributes) as attributes of an association class. Since the interaction does not yet concern identifiable product things, this is a binary association class. Fig. D.1 shows a model that expresses this interpretation.

Note that there is no class representing the products in this model. This is because the products are not relevant in this interpretation. The products possess as yet no mutual properties either with the customer (since she ordered anonymous product types), nor with the employee that received the order. Thus, details about the quantity and the price of each product type to be supplied are mutual properties between the employee and the customer. These reflect the agreement that exists between the customer and employee for the employee to deliver the required quantities at a certain date and the customer to pay the agreed upon price.

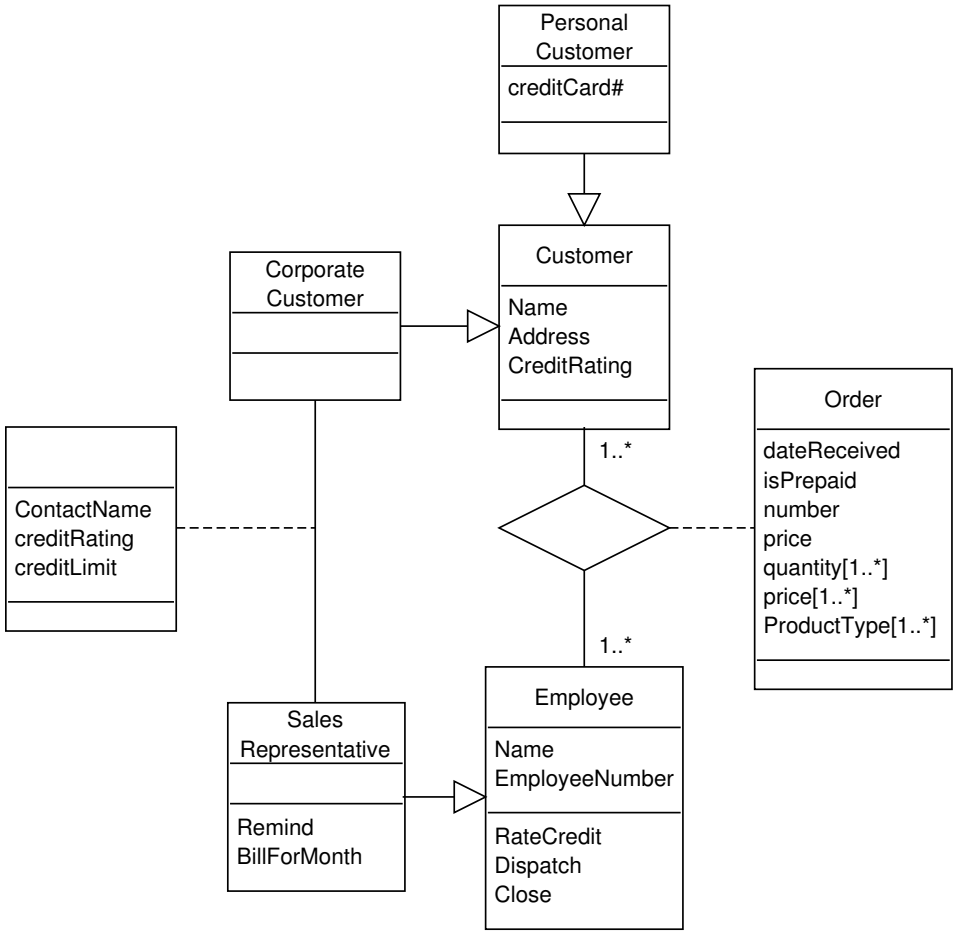


Figure D.1: Example class diagram with ontological semantics

Note that the operations 'Dispatch' and 'Close' are modelled with the employee, and the operations 'Remind' and 'BillForMonth' are with the sales representative. By rule 4 (changes in mutual properties as operations) these are therefore operations of the employees and the sales representatives, not of the order, as it is they that do the dispatching, the reminding and the billing.

At a later stage in the order fulfillment process, e.g. when the employee takes the required products off the shelf to assemble the shipment, the product items become identifiable. When e.g. the employee interacts with the product to assemble the shipment, this interaction gives rise to mutual properties between the products, the employee and the customer. As an example, there exist mutual properties that a particular product item is reserved or assembled or shipped etc.

This situation is somewhat similar to first interpretation in Section 9.2, where we assume that not late in the order-fulfillment process, but immediately when ordering, the product items are identifiable, e.g. because the customer picks out items in a showroom or reserves specific items through the employee for later delivery etc.

Appendix E

Case Study Interviews

This chapter contains the transcribed interviews that were conducted as part of the case study. In the interest of readability, the transcripts have been cleaned in the sense that sentence structure was made readable and sentence fragments were joined, etc. The choice of words and the sequence of expressions remains unchanged. The following sections contain verbatim responses of the subjects. Notes and questions asked by the interviewer are shown as *emphasized* to distinguish them from the interviewee responses.

E.1 Lead Analyst Interview (LF)

The main entities in the analysis domain are high school students. A distinction is made between local British Columbia (BC) high school students and other high school students. BC high school students are either students in grade 10, grade 11 or grade 12. The students attend or have attended BC high schools. A student takes a number of courses with a BC high school and receives a mark for each of these courses. The admission rules for the university are based on course mark averages for grade 12 courses, while successfully completed grade 11 courses are required for admission but the mark is not relevant. Students may take the same course more than once, in which case the highest mark is the one relevant for the admission criteria.

The curriculum for BC high schools is defined by the provincial ministry of education. However, some schools also offer courses from the international baccalaureate (IB) curriculum, which are recognized in the admission

requirements. There is a mapping between IB courses and BC courses. Students must choose one of the programs offered at the university in order to request an admittance assessment.

How are students identified? What are relevant properties of students?

High school students are assigned a student number and a PIN once they have applied to a university program. This occurs during the normal course of any application, before any decisions regarding admissions are made.

Another method which is currently used is that students interact with the ministry of education through a web site and provide identifying information. This information is then forwarded by the ministry to the university, once students express an interest in a program.

The university itself collects identifying information from students, again through a website. This is termed prospect tracking. Here, students provide identifying information and choose a login and password which identifies them to the university. This system is integrated with another university identity system, the campus wide login (CWL) project. It is planned that the CWL will eventually supersede the student number as the primary identification.

Who maintains the IB to BC translation table?

Course translation tables are maintained and owned by the admissions committee which reports to the senate. For post-secondary institutions, transfer credit is assessed through transfer credit tables. For secondary education institutions, this is done through a rule based system. For each combination of jurisdiction (e.g. Canadian provinces, Internationally recognized programs, etc.) and program there exists a set of rules to determine the equivalent BC high school course and grade.

What distinguishes grade 10,11,12 students, the number of credits or type of courses taken?

Regardless of the number of secondary education credits of a student, a student is determined to be grade 10, 11 or 12 depending on the type of courses completed successfully.

Who determines the admission requirements for programs?

The admission requirements are established by the admissions committee, reporting to the university senate. However, actual admissions procedures are based on a competitive average. Based on the number of positions

available, the university selects the best students, once all applications are submitted.

How are high schools identified?

High schools are internally identified by numeric codes. However, the actual identifying information is the name and address of the high school.

How are courses identified?

Courses are identified by the province or jurisdiction where the course was taken, the course type, a course description and, for IS purposes, an EDI (electronic data interchange) code.

What different states/status can students be in? E.g. interested, applied, accepted, admitted, etc. How are these defined?

In the first phase of the project, the university deals with applicants only. It is planned to extend this to also include prospects, i.e. students that have expressed an interest. Prospects may become applicants, which, depending on their grades, may be admissable. Once a student has applied, an admissions officer makes an admissions decision. Thus, the student may be admitted, refused, found to be not eligible, or not yet evaluated.

How are mixed students handled, students that e.g. receive grade 11 in Alberta and grade 12 in BC? Or grade 11 in BC and grade 12 in Alberta?

Currently, these students are processed manually on a per case basis and it is not planned to apply automatic rule based admissions criteria to these types of students.

What documents are involved? Does documentation (or grades) have a status? E.g. provisional, final, documented, official, etc?

Grades are entered through the web-site, these grades are un-official. For desirable students, the admissions office contacts the high school to confirm these grades.

For BC, Alberta and Ontario high schools, grades will be electronically received from the respective ministries of education, while for all other students, the grades are received as official transcripts sent by the respective schools. Since Canadian transcripts are received directly from the High schools, no certification or other credentials are needed for such grades.

What about TOEFL scores?

For the purpose of collecting TOEFL scores, the university interacts elec-

tronically with the TOEFL company. The identifying student information for this purpose is the full name and the birth date of a student. TOEFL scores are received electronically.

E.2 Admissions Officer (RP)

For purposes of admissions, three distinct groups of students exist, based on their secondary school location: BC high school students, students at high schools in other provinces of Canada and international students. Because all three kinds of students have a different background and require different amounts of time, a different admissions process is required.

The university receives interim (in May) and final (in July) grades of students from BC high schools in an electronic form. Early interim grades can be reported by the student to the university through the university's web site. Students would not receive any immediate response from the university. Formerly, early interim information was reported by student counsellors in BC high school to the university by mail or fax. Early interim grades reported by students are verified by the university by contacting by fax or mail the student counsellors at the corresponding high school. Based on verified interim grades, the university makes a decision whether to offer the student early admission to the requested program.

Students in an international baccalaureate (IB) program have their anticipated grades forwarded by the IB co-ordinator at their school to the university by mail. Based on these grades, the university makes a decision whether to offer the student early admission to the requested program.

For all other Canadian provinces, students can have their schools supply their official first term grades by mail. Based on these first term grades, the university may decide to offer admission. Final results, second term grades, are also submitted by mail by the student's school.

Based on a received application, the university determines what further information and which reports are needed. These are then ordered from the respective school, which in turn sends them to the university.

TOEFL scores are gathered electronically from the TOEFL company. Every student who undergoes the TOEFL examination can indicate institution codes to which their scores should be made available. The university gathers student scores according to their institution code. These scores are

then manually matched with student application records at the university, as not all students indicating the university as possible recipient also apply to the university. The same process takes place with respect to SAT scores, which may be available from some students of US high schools. Generally, US high school students are considered based on their academic record and the process is similar to that for students of other Canadian provinces.

Early admission decisions are based on admission criteria for each undergraduate program that are expressed as percentage cutoff points. The grade scales and cutoff points for admission are determined by the individual faculties offering the programs. These are determined according to the number of government funded seats. The same criteria are then applied to international students as well, even though they do not take up funded seats. For the evaluation of international students, there exist a number of publications by different companies and associations which are used to determine the grade equivalencies. The university uses this information initially, but then, based on past experience, an admissions officer makes the final determination of the equivalent grade. International students require their school to supply original documents as well as translations of those documents. These may either be done directly by the school, by another government body or a certified translator.

When a student submits early interim grades to the university by mail or fax, these are sent to an admissions officer for evaluation. When a student submits early interim grades to the university by means of a website, this information becomes available to admissions officers in a database. The admissions officer then manually selects students from the database to process. Students at this stage fall into one of three groups, unconditionally acceptable, conditionally acceptable or on hold.

Students are not rejected based on their early interim information, but rather placed in the hold group if their grades do not satisfy the admission criteria for the program applied to. For the students for which their grades indicate that they are unconditionally or conditionally acceptable, the admissions officer then contacts the school by fax to verify the grades reported by the student. The school sends verification by fax back to the admissions officer.

Students are unconditionally acceptable if they have good final grades or extremely high interim grades. Students are conditionally accepted if they possess high interim grades, but the possibility exists that these can slip below the determined admission cutoff point. For these students, an

admissions officer checks subsequent grades as they become available from the school or the ministry of education. Students who are not conditionally or unconditionally accepted remain on hold. These are students whose current grades indicate they are below the cutoff point, but where the admissions officer determines a possibility that either the grades will increase or the admissions cutoff will drop to include those students. Students are only rejected based on final grades once they have formally submitted an application. Self-reporting bad grades will not result in that student being rejected. Rejection occurs either based on poor grades or on lacking prerequisite courses for the particular program applied for.

In May, the provincial ministry of education makes grades available in electronic format to the university. These grades are still interim grades, not final grades. The university then calculates the appropriate grade average based on the courses required for the program applied for by the student. For students who have previously been unconditionally accepted, the university determines whether these students become eligible for scholarships. Students who have been admitted conditionally, may either be offered unconditional acceptance, remain conditionally accepted or be placed on hold. However, this process can vary depending on the faculty that the student applied to, as some faculties offer broader based admission (BBA) criteria. These applications are forwarded to a faculty admissions officer who will then contact the student's school for more information. The faculty admissions officer will then make an admission decision based on the BBA criteria and communicate this decision to the university admissions officer, who will communicate the decision to the student.

Once final grades become available in July from the ministry of education, students in both the hold group and the conditional acceptance group are again examined by admissions officers. They will either be offered acceptance or be rejected.

The averages required for admission are based on enrollment and may change. Generally they drop as students decline offers by the university. To determine these admission criteria, the admissions department tracks the number of applicants for each program against the number of offers and the numbers of registrations by students in that program. The aim is to exactly fill the number of funded seats. Above target seats are unfunded by the government and if the target is not reached, the government will adjust the number of funded seats for the next academic year downwards. The adjustment of admission criteria is based on past and current statistics and

estimates by the admissions personnel.

E.3 Student Recruiter (AMJ)

The university's student recruitment activies fall into two categories: Proactive recruitment and a reactive recruitment and advising side. The proactive side involves student recruiters visiting high schools, colleges and career fairs. This serves two purposes, to promote and provide information about the university and second, to identify selected students for targeted recruitment activities. Student recruiters cover about 85% of BC high schools as well as high schools in Alberta, Saskatchewan, Manitoba, Ontario, Quebec and Nova Scotia, beginning in September and continuing throughout December. From January to March the focus is on programs on the university campus. These are information sessions for high school and college students and their parents, faculty days, and a number of other programs.

On the reactive side, student recruiters communicate with students that were met during high school visits. Recruiters give out personal contact information during these visits and are generally contacted by email and answer questions related to applications, admissions and housing, among other areas. Currently the university has no information system for managing or tracking such contacts.

Recruiters also visit local colleges once a month for a full day of student advising and recruiting. Emails that are sent to the general student inquiry address of the university are also forwarded to student advisors. Email volume is from 60 to 80 messages daily to about 200 to 300 per day during peak times. Besides email and telephone, recruiters and advisors also offer drop-in and booked appointments on campus.

The BC secondary school and college liasion program (BCSSCLP) is a program that is used by all universities and colleges in BC to co-ordinate their high school visits, in such a way that at least three institutions are at a given high-school for a recruitment day. Students attending these events are mainly grade 12 students although, depending on the particular high school, grade 11 students may attend.

Increasingly, the university understands high school counsellors and parents as their target audience for recruitment activities. Any high school has generally more than one counsellor. Some schools have counsellors for a specific grade, other schools rotate cousellors so that a counsellor stays

with the same group of students from grade 9 through grade 12, yet other high schools assign students to counsellors alphabetically. It may be difficult for recruiters to find the appropriate counsellor. In those cases, the BCSS-CLP maintains an online database for high school counsellors to self-update current information about their school for the purposes of that program. Recruiters may also suggest possible dates for visits using this information system. Contact with out-of-province high schools is generally established through the school's secretarial staff who then directs the recruiter to the appropriate counsellor(s).

There is no accurate data about applicants by geographical area and by high school to see whether recruiting activities are successful. There is no information available about housing applicants who may need help or reminders. Recruiters also need more accurate information about grades to assess which areas and which individuals to target for scholarships. The university plans on developing a customer relationship management (CRM) IS that would help channel, manage and co-ordinate all these activities. However, currently all communication with students is done by email which is not shared or centrally managed in the university.

E.4 Students

The recruiting process begins for high school students in November for the following academic year. At this time, either the student counsellors or the university recruiters coming to high schools make the students aware of the university, it's programs and the application procedure. Both, counsellors and recruiters, can provide students with paper forms or direct them to the university's online application system and other resources at the university, such as student advisors.

High school students have two ways of applying to the university. The first is paper-based, by filling out and sending in the required application forms. The second way is through a government web-based service called PASBC. This site allows the student to provide personal information and use this information to fill out electronic applications to all post-secondary institutions in the province. For either kind of application, the university charges an application fee, which is due at the time of application. Paper-based applications are generally accompanied by cheques or credit-card authorizations, while PASBC is able to process credit card payments electron-

ically. Unless and until the application fee is received by the university, no processing takes place.

Upon applying, the student is requested to arrange for official transcripts from the high school to be sent to the university. However, the university will independently request the transcripts from the high school's student counsellor, so that the application process, from the point of view of the applicant, is over. It is the high school's obligation to keep sending transcripts as they become available.

Upon application, the applicant is provided with a student number and password to enable the use of online services at the university. This information is sent either by email or by regular mail, depending on the chosen preference of the student. The applicant can use this information in order to apply for housing, scholarships, etc., but also to check the status of the application using a web-based service. The status may be either "pending", "conditionally accepted" or "unconditionally accepted". A conditional acceptance is based on interim information that the student's high school counsellor may have submitted, such as mid-term grades, etc. which are unofficial. The PASBC system does not support any grade submission, this must be done manually by the high schools. The university online services also provide information about required documents that have not been received by the university.

The local high school student provides two kinds of grades. The high school transcripts show information about the standing of students in various courses. Final grades and transcripts for in-class standings become available in January and in May. In addition to this, high school students are required to take province-wide examinations in a number of subjects. These exams are taken in January, May and August and grades become available in February, June and August. The August examinations are generally only for those students wishing to repeat an exam. The Ministry of Education maintains the grades and transmits them to the university.

Depending on the program applied for, a student may be unconditionally accepted as early as February, if that program's entrance requirements are fulfilled by first term courses and first term provincial examinations. Otherwise, students may be unconditionally admitted in May. Due to the nature of a sliding admittance scale based on the grade-point-average (GPA), some students may not be admitted or rejected until August, when the GPA requirement has been lowered sufficiently. The admissions process ends in August, with the final rejection or unconditional acceptance of all applicants.

The university is currently implementing a web-based information system that allows the applicant to submit their own estimated grades and check whether such grades would fulfill the admissions requirements. For this, the applicants require the student number and password information sent as a result of their application. Students have the ability to change and alter their estimated grades as often as they like and re-check their admissability. The estimated grades may be submitted to the university once, after which they cannot be altered. At this point, these grades are available to the university admissions officers.

E.5 Discussion with Project Lead (LF)

This section details the final interview with the project lead LF. The results of the analysis (Appendix F) were provided to LF with the request to read and assess with respect to specific criteria such as readability, ease of understanding, correctness, completeness and clarity.

LF began the interview with the comment that he found the diagrams interesting and asked about whether any strict rules were applied to it. The interviewer explained rule 1 and the motivation behind the rule. LF commented that from personal experience, "anything is an object, they can be mathematical objects, syntactic objects, anything". He commented on the way that a course was modelled in the class diagrams, as an association class: "right, there is something going on. It's normally difficult to model a course object, because it is a relationship". He commented further that one has to ask "what do you mean by a course? The curriculum, the interaction, the grade?". He agreed that rules may be used to address this problem and force the modellers to think deeper about what they are modelling.

LF commented that the final exam interaction with the ministry is not accidental, the course that students take is part of the ministry of education curriculum. Similarly the international bacccalaurete program has a curriculum and there are international baccalaureate final exams.

How would you model a curriculum?

"There is a calendar which prescribes rules for the courses and which also prescribes content of individual courses." He brought up the example of the university where the senate defines the curriculum but the faculties apply the general guidlines and this then "becomes the real practical curriculum".

LF commented that "interesting in the description are the different applicant states. You put in BBA admissable. That brings up the fundamental question: What are admission rules? In practical terms we tend to model them as anything else. But presumably needs more thought to distinguish different kinds of objects. A rule is not the same kind of object as a brick. BBA is not a state a state of an applicant but is a requirement or more support for the application, e.g. work experience, community work, etc." As an example, LF mentioned the faculty of medicine where all these other factors are also evaluated numerically while for the MBA program this is less formal. In summary, LF suggested that BBA is an extension to the core set of admission rules.

Which set of object models do you think provides a better introduction to somebody new to the project?

"Yours that you have are arguably better as an initial introduction whereas ours have all sorts of stuff around that is assumed but not modelled. Yours presents a much clearer picture for somebody who goes into it as to the basic relationships. But at what point do you slip from the object to the programmable model? Your sequence (diagrams) are very good to show what actually happens. I would call them conceptual models whereas we use sequence diagrams to show method calls. Your're right when you say that sequence diagrams are bad at showing multiple interaction."

Which do you think would be better suited for database design?

"Ours is closer but we've deliberately avoided data modelling. There's a real danger seeing object modelling as data modelling or relational modelling, as rows and foreign keys.

I find the modelling of the real world gets quite tricky and problematic. Modelling of non-substantial objects happens quite easily and naturally because you don't have to worry about empirical reality. The real power of this (the new models) is the progression from describing your buisness to program is quite seamless.

That diagram (old model) is just punched out, almost reverse engineered."

"These are definitely states. Are they states of the applicant or the application? Do you see an application as a paper object?"

You would model an application as an object?

"It's a problematic one for us. It just doesn't do anything, it's a state-

ment of intention."

"There is a more generic, archetypical diagram, which should be all about evaluating requirements of which all, SAT, TOEFL, BC High school are all specific examples.

This raises a key issue for moving the project ahead: Do we model TOEFL and SAT as separate objects? Or do we have a single class 'test' that can be extended by TOEFL and or SAT? Where does one set the boundary to these real world things, objects? Same with BC high school students, who is an object. This goes back to an early argument in the project group."

E.6 Discussion with Lead Developer (CH)

CH is the lead developer of the project. At the time of this interview, he had been with the organization for three years. He was not involved in the Business Process Reegineering (BPR) project that preceded the project under study, nor was he involved in the organizational analysis of the requirements of the project. His responsibilities in the project are focussed on design and implementation aspects, but as lead developer he is aware of the business and organizational requirements.

CH has extensive UML knowledge and experience from prior projects but noted that in this project and in this organization, UML has only been introduced about a year previous to the time of the interview, about the beginning of the particular project.

"The diagrams show a lot of distinction between different classes of students, while the project team model employs attributes to make those distinctions, to flag different types of students. The model was developed as an analysis model, but the modellers may have unconsciously seen the model's purpose in terms of guidance for IS design, therefore choosing a way of modelling which makes implementation easier by requiring less coding.

"Inheritance also gets shied away from in the project team model. This may be because it is initially confusing to developers, they may not understand it or use it properly."

What model would be better for comprehension for a new project team member?

"Yours (the independent model) is more comprehensive, for example in the relationships. It shows a better "big picture view" of how it fits together. It would undoubtedly have benefits when bringing on different staff. For our model, we relied a lot on assumptions that were never written down in the model.

It [the independently developed model] shows a lot of stuff that is not implementation related, e.g. the ministry of education. That is needed but not we don't need to do anything with it, we only need to know of its existence. "

Do you think the models could serve as starting points for design?

"There are more requirements needed. Assuming that the operations are modelled, it would be easy to start coding and start on your development process, whatever that process is. I don't see any reason why you couldn't just take these and run with them. The sequence diagrams are very useful. The class diagram in and by itself does not explain much. One thing that the tools we use don't allow is they cannot generate methods from sequence diagrams. You have the call stack from the sequence diagram and the messages, you should be able to put them into methods automatically."

LF suggests that sequence diagrams are very unusual in the analysis phase.

"There are times when we do use them. There was time when we mapped it all out. That included the infrastructure level as part of the design."

Do you think the sequecne diagrams are helpful in understanding the organization?

"They don't get done as official project documents. When developers meet, two thirds of them will use inofficial sequence diagrams that never get written down, especially for the infrastructure aspects. That makes things much clearer.

One of the reasons why they're not used for domain modelling in this project is the small group size. We're only four people involved, so everybody was aware what was happening. If people weren't deeply involved it would be very helpful for the designers.

The sequence diagram shows what the process is, but it also shows parts that we do not inherently do, e.g. the admissions officer would not have to manually send aceptance notices anymore, that is something that we're trying to automate."

Are there any constructs in the model that are too complex or too complicated?

"The large diamond sort of throws me off. Don't know directly what we're implementing there. That could be a function of the tools that we're using, that they're not implementing the full UML spec.

When you start getting a variety of lines crossing, it becomes tough to package it up, to really understand the relationships quickly. It then takes a while to understand and pull them apart."

Your domain model has no association classes and fewer classes with more attributes.

"That is probably based implicitly on the existing database. Mapping to relational databases is easier with flatter classes, easier than taking apart object structures.

Our model wasn't explicitly based on the databasse but the fact that people were familiar with the database structure, this influenced things. The fact that there are not many inherent business rules makes it easy to model this as a flat database. Implementation wise it is of course faster, too.

The lack of association classes is probably just a poor use of tool, really."

Do you think it would have made a difference to model association classes as associations instead?

"Visually, this shows better the association. It gives you a better sense of the relationships.

Your model is a different view on the problem space from that what we implemented, but there's no obvious flaw. From a pure OO sense, encapsulation point, yours is probably more correct, certainly more comprehensive."

Is it comprehensive because it contains parts not implemented or is it also comprehensive in the parts that are implemented?

"Certainly more comprehensive there, but even in the smaller, there's a somewhat simpler, more elegant view in a few cases.

Your class names are clearer, more descriptive.

In our project, the requirements are pretty fluid, so there's not that much time spent on analysis documentation."

The interviewer explained the background of the case study to CH.

"The rules seem to force rigorous design and force you to ask the questions, if not necessarily to follow them. They are certainly the right questions to ask.

Such rules would hve helped in our group. The rules would tell whether a model is good and can help answer some questions. They seemed like a lot of valid questions to ask. "

Appendix F

Case Study Analysis

This appendix provides the details of the business and organizational analysis and conceptual modelling carried out as part of the case study (Chapter 10). It begins by developing the static structure model and the interaction model. In a second step, operations for the various classes are added to the model.

F.1 Static Structure and Interactions

As described in Chapter 10, the approach taken is largely based on an analysis of the interactions. Thus, the two main diagram types are sequence diagrams and class diagrams, with supporting state charts when applicable. These diagrams cover both structural and behavioural aspects of the domain and can be compared to the original class diagram developed by the project team.

The analysis begins with a focus on local high school students and is then widened to include out-of-province students, US students and international students, followed by an abstraction of common features. We abstract from communication media unless they are relevant to the case. They should be modelled as they are things which interact through mutual properties with the communicating things. E.g. a phone line between two people has properties with both people and serves as intermediate object. However, neither phone lines, nor email or postal mail is modelled explicitly (see rule 33 in Sec. 6.1.1).

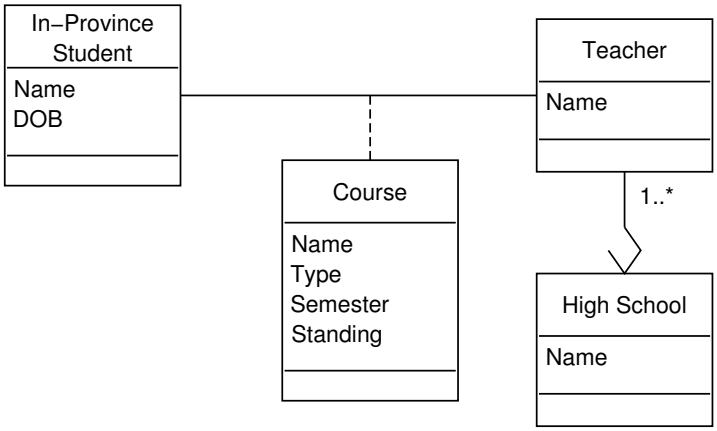


Figure F.1: Student and teacher classes

Local high school students are substantial things in the ontological sense and should be modelled as objects and represented by a class in the class diagram. Students interact with teachers of a high school throughout a course. As a result of this interaction, the students acquire mutual properties, their in-class standing for that particular course. A course is not a substantial thing in the sense of rule 1 and can therefore not be represented by a class in the class diagram. Instead, a course is an interaction (strictly: a set of interactions) that gives rise to a bundle of properties (rules 2, 3, 5). Figure F.1 shows the class diagram and figure F.2 shows the sequence diagram depicting the interaction.

By rules 8 and 10 there must exist uniquely identifying attributes. For high school students, these are the name and date of birth (DOB), while high school teachers are identified by their name and their high school. High schools in turn are identified by their name. The current class diagram may possibly violate rule 13, according to which each aggregate must have at least two parts. As the diagram develops, this will be addressed.

In the next step of the process, students apply to the university. The university is interpreted as a composite thing, made up of faculties, which in turn are composed of staff etc. Since we are concerned with a single specific university, it is represented as an object instead of a class. As a result of the application interaction and the university's response (either by mail or e-mail), students and the university gain mutual properties represented by an

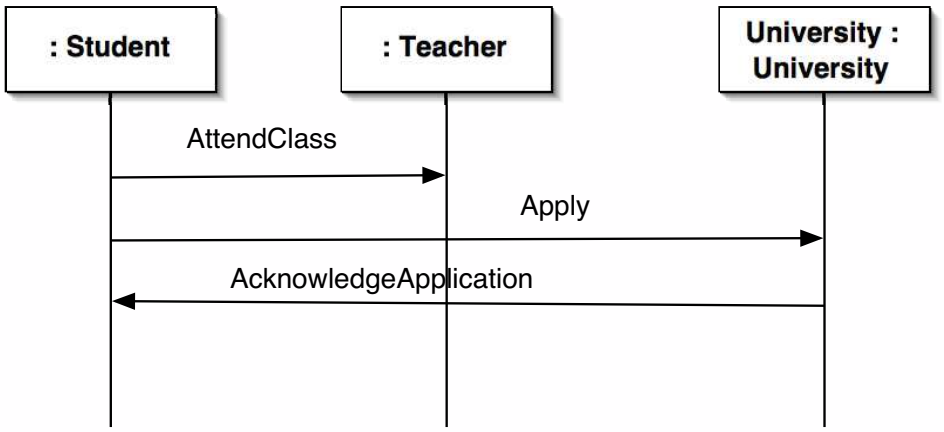


Figure F.2: Student and teacher interaction

association class. These properties include the program applied to, and also the student number and password issued by the university. Importantly, upon application students acquire the property of a fee balance with the university which represents such items as application fees, tuition fees, etc. Since at that point some students will possess these additional properties and others will not, rules 12, 11 and 19 suggest that these students should be modelled as a subclass, that of applicants (see Fig. F.3).

Students and applicants also interact with the Ministry of Education (MoE), by writing provincial examinations. This interaction leads to sets of mutual properties of the ministry and the student. We allow for the possibility that a student takes more than one provincial examination and also attends more than one course with the same high school teacher. As UML, in our ontological interpretation, is very limited in providing ways to define complex data structures, we assume that the i-th subject has been taken in the i-th semester with the i-th standing. Similarly, the student has taken the j-th subject provincial examination in the j-th semester with the j-th grade. The interaction and the resulting changes to the class diagram are depicted in Figs. F.5 and F.6. The sequence diagram also indicates that the stimuli corresponding to the two messages with actions "AttendClass" and "ProvincialExamination" may occur more than once.

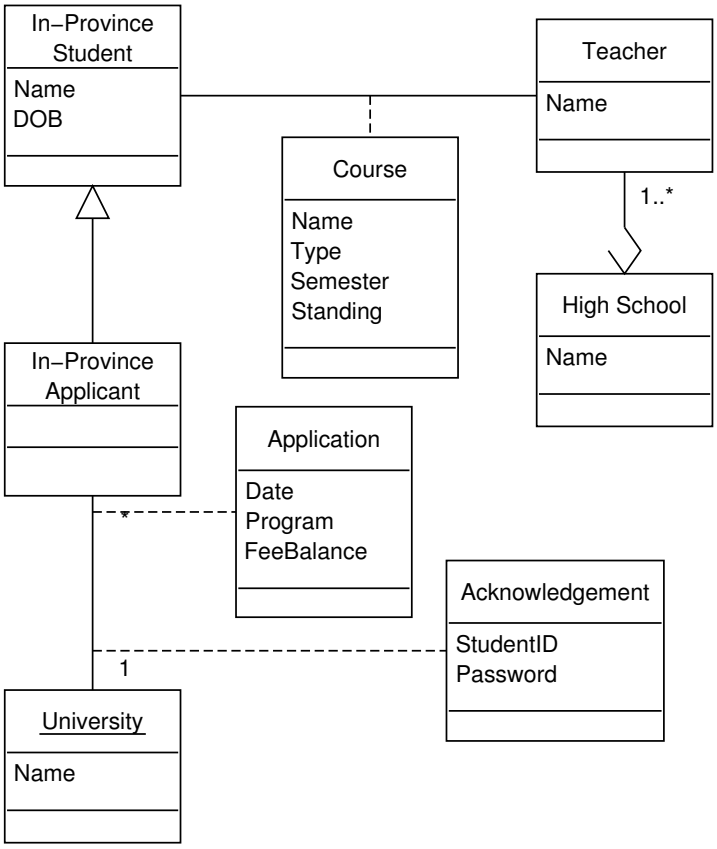


Figure F.3: Applicant and university classes

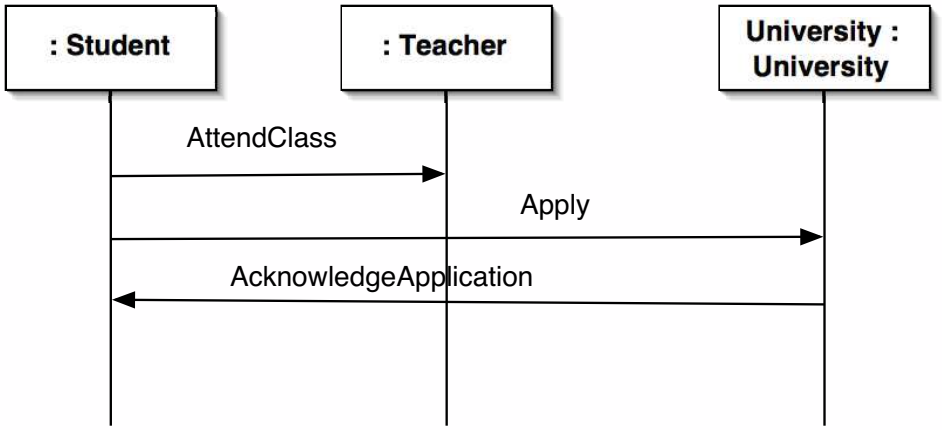


Figure F.4: Applying to the university

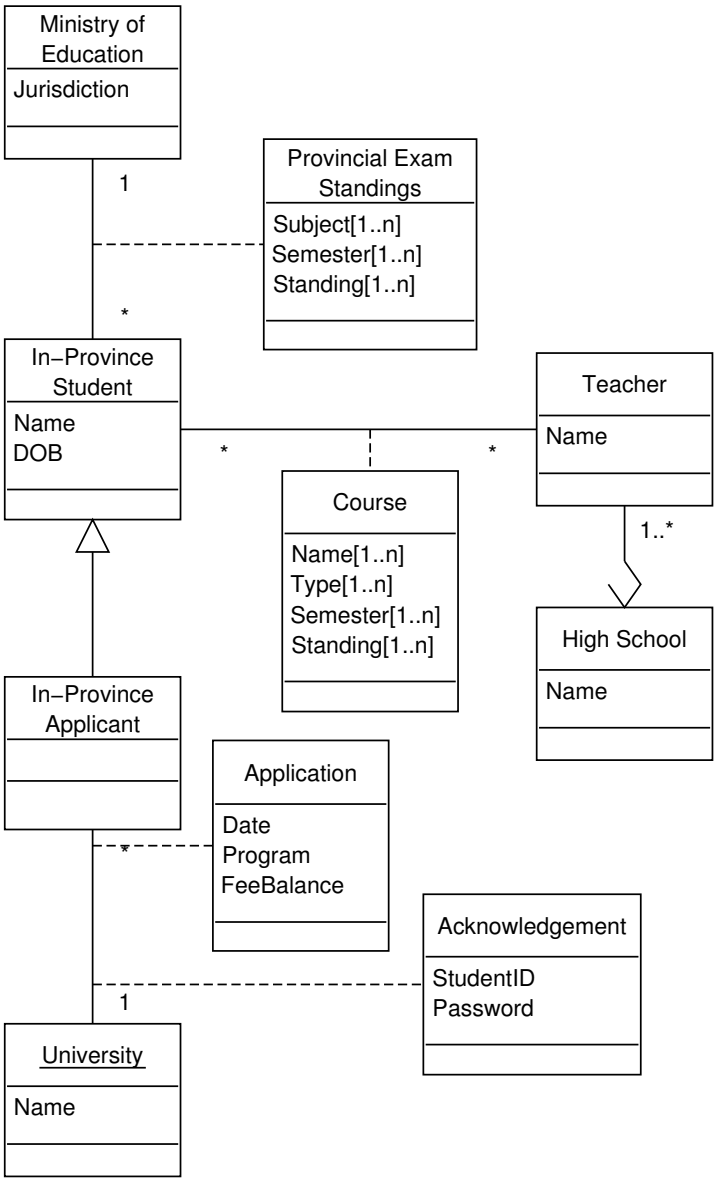


Figure F.5: Ministry of Education class

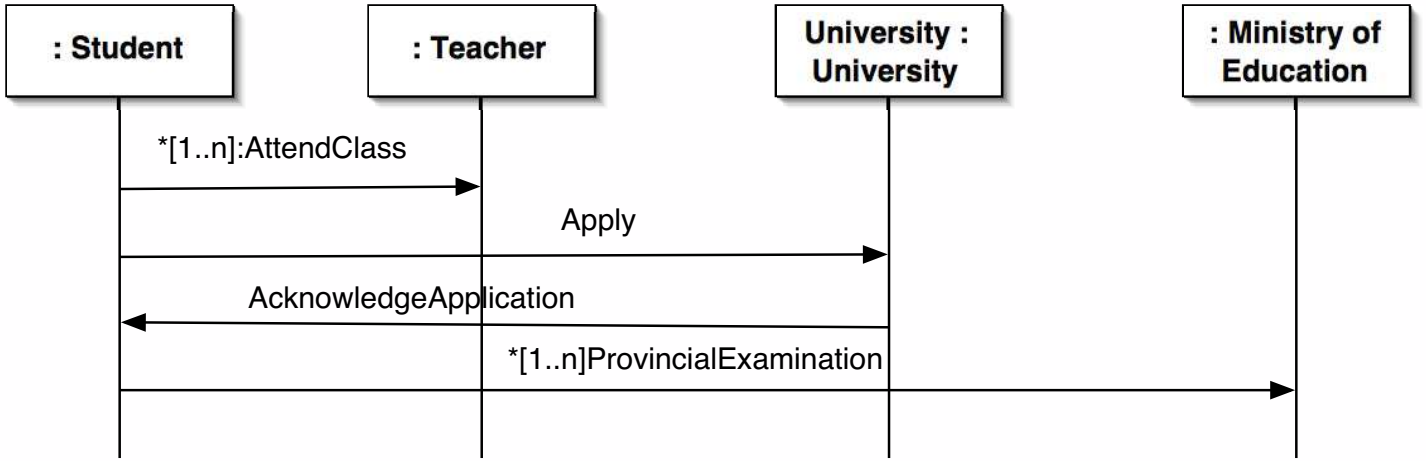


Figure F.6: Writing provincial examinations

The applicants may then request their high school counsellors to send interim grades as well as final transcripts to the university. Information about provincial examinations is forwarded by the Ministry of Education to the university automatically, without request. Both interactions lead to mutual properties between the applicant, the university and the student counsellor or the Ministry of Education. The resulting properties are represented as attributes of association classes. They represent the grades that are reported and may, but should not, differ from the grades represented by mutual properties between students and teachers or students and the Ministry of Education. When a high school does not submit transcripts for a student, an admissions officer will request them from the high school's counsellor. Note that the reported grades are mutual properties also of the student. The students should not be represented as attributes because of corollary 1, therefore students must be participants in the associations. Three of the described interactions, the submitted transcripts, submitted interim grades and reported provincial grades, lead to mutual properties between the student, the university and the high school's counsellors or the ministry. Two others, the request to submit transcripts and the requests to submit interim grades, do not lead to such mutual properties. Provincial grades are submitted up to three times (corresponding to January, May and August grades) while transcripts are submitted twice (for the first and second term). Interim grades may be sent by the high school student counsellor multiple times. When students self-report grades through the online service, the grades become mutual properties of the university and the applicant. A university admissions officer then requests the high school student counsellor to verify the grades. The verification message sent by the counsellor gives rise to verified reported grades, which are treated as interim grades, as opposed to final transcripts.

These interactions are shown in the sequence diagram in Fig. F.7. UML requires sequence information for both sequence diagrams as well as collaboration diagrams. Therefore, there exists no way to indicate in the diagram that the sequence of the messages/stimuli is not relevant and may vary. Hence, Fig. F.7 shows only one possible sequence of messages. Figures F.8, F.9 and F.10 show class diagrams with the associations and classes participating and arising out of the interactions.

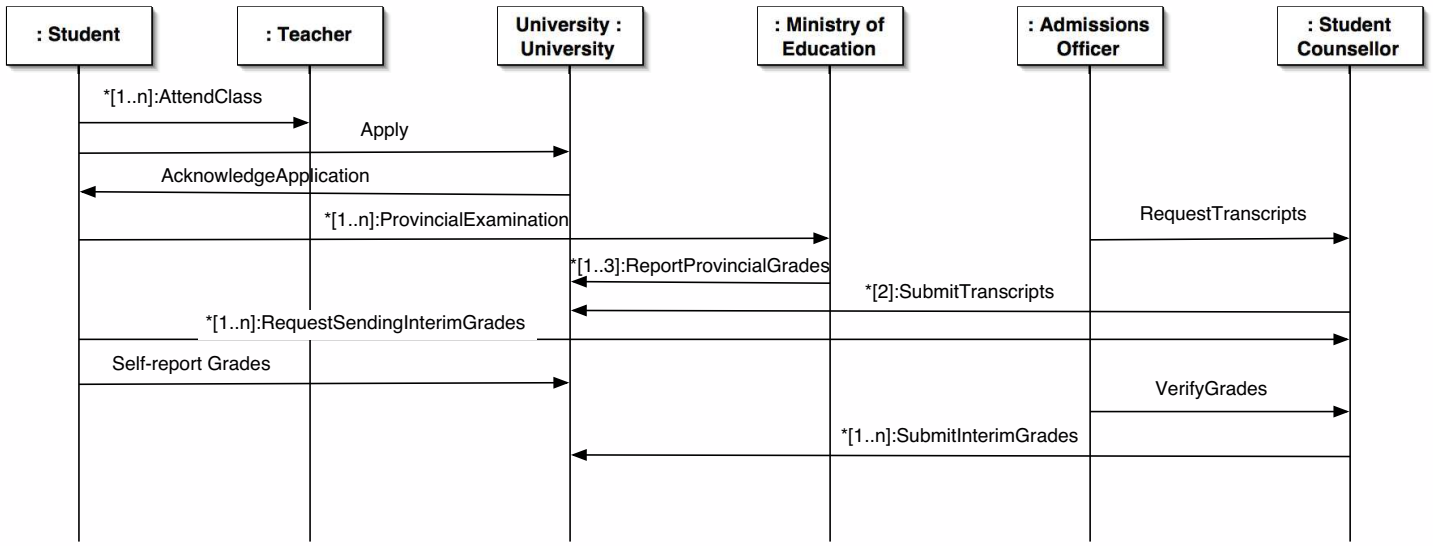


Figure F.7: Submitting grades

Once the marks are received and the payment has been received (shown by the value of the outstanding balance, a mutual property of the applicant and the university), the applications are assessed and their status determined. This is done repeatedly by an admissions officer, when new information becomes available. The admission standards are set by the faculty which offers the program and are therefore properties of the particular faculty. Requirements are sliding requirements in the sense that they depend on the number of available places for a particular program. Thus, initial requirements are specified by the faculty, but actual requirements are determined as the admissions process continues, based on the number of applicants, the number of accepted offers, the number of withdrawals of applications, etc. The faculties are modelled as a composite object which is part of the university in Fig. F.11. An applicant's states change when she is re-evaluated based on the current requirements.

It may appear correct to assign a state to students, based on their outstanding balance, their grades etc., e.g (see Fig. F.12)

- Pending (Fee not paid)
- Hold (Incomplete grades, some grades below the requirements)
- Conditionally accepted (Incomplete grades, current grades meeting requirements, some required grades not known)
- Unconditionally accepted (Complete grades, all grades meet all requirements)
- Not Admissible (Complete grades, some grades below the requirements)
- BBA Admissible (For faculties with programs with broader based admission requirements)

As a result of rules 21 (states are defined in terms of attributes of the object), 25 (quantitative behaviour is represented in a state chart), 26 (state transitions correspond to operations of the object) and corollary 14 (all operations change the value of an attribute or association class attribute), these states *cannot* be states of the applicant. Rule 21 prevents defining those states with respect to the admission requirements, which are not properties of the applicant. Rules 25 and 26 would require operations of the applicant to change the state. However, it is clear that the applicant does nothing

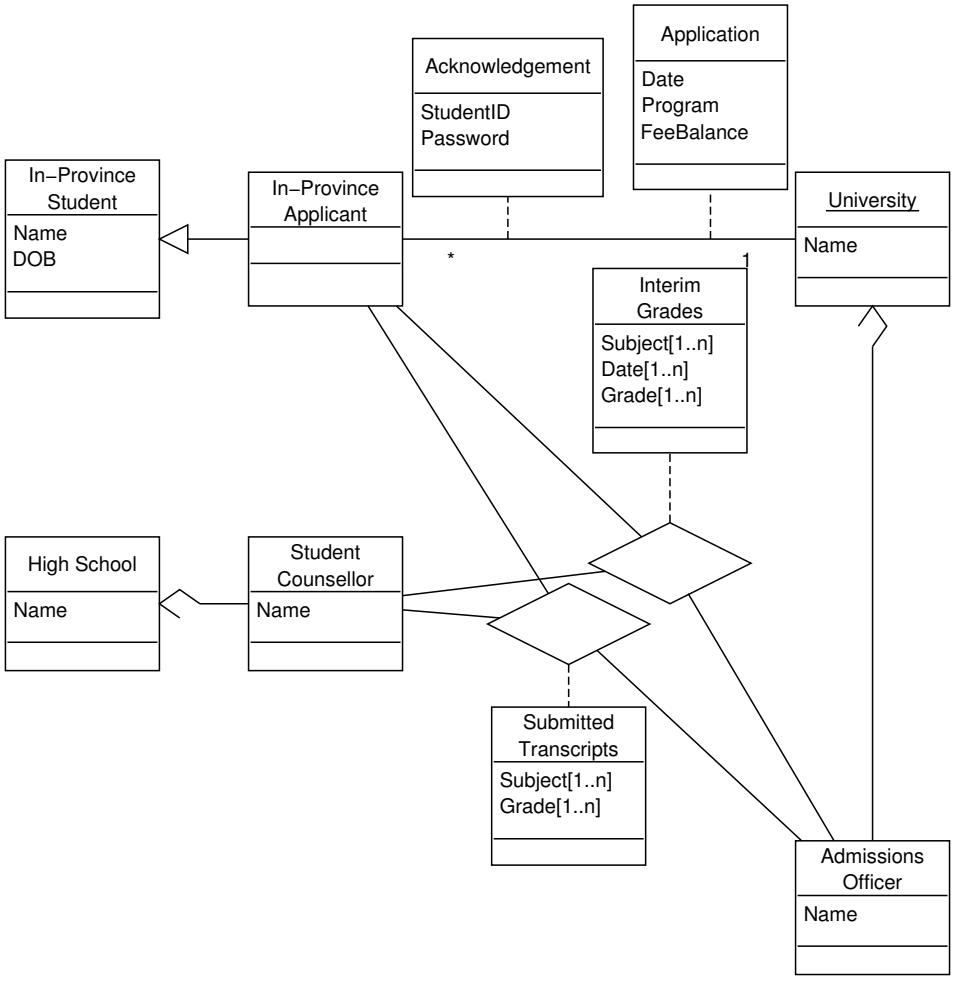


Figure F.8: Submitted transcripts

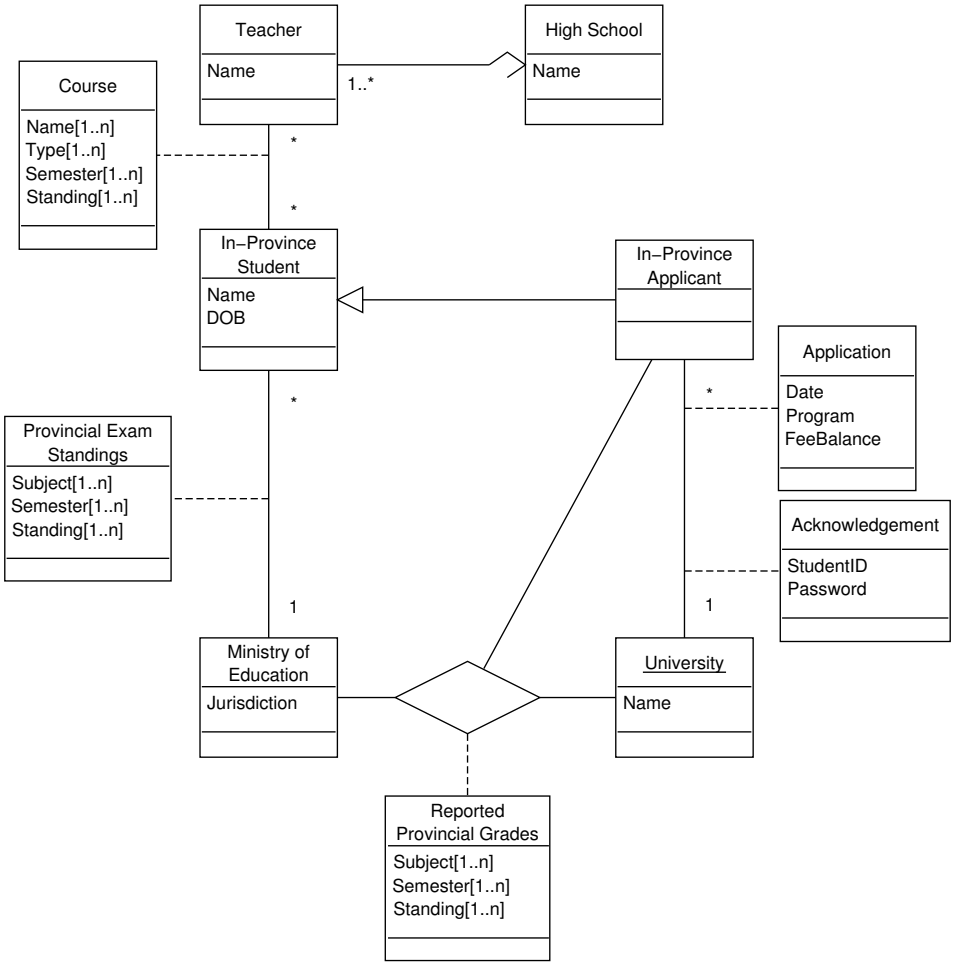


Figure F.9: Submitted provincial grades

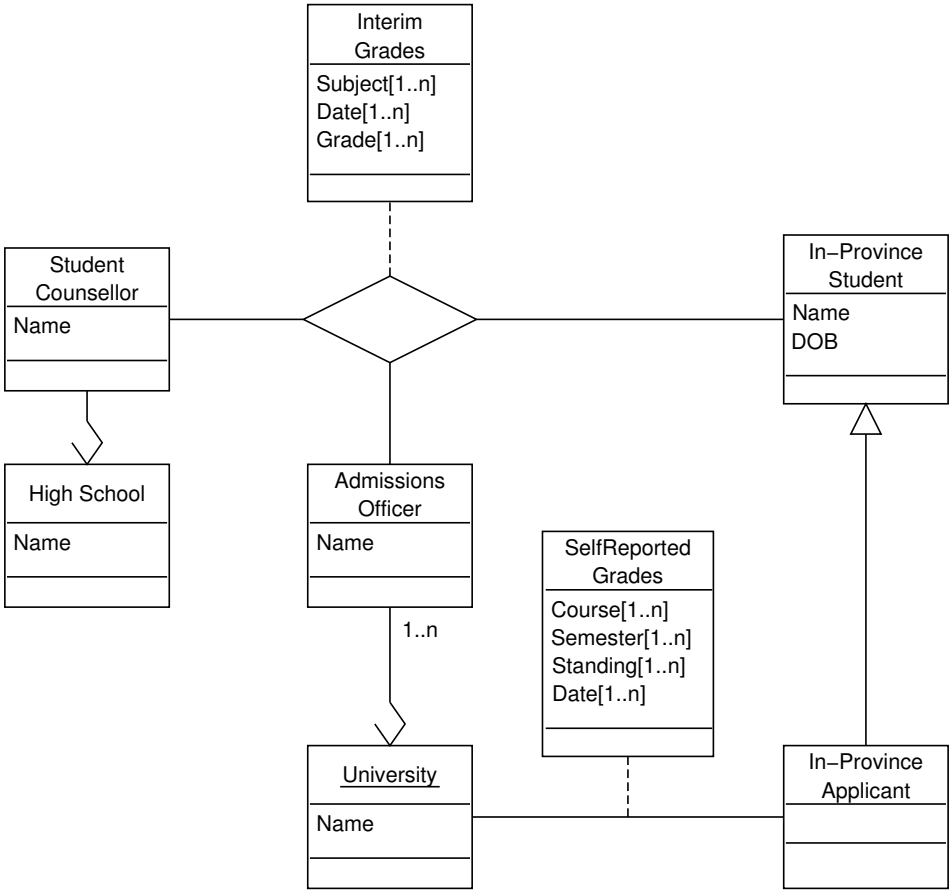


Figure F.10: Self-reported grades

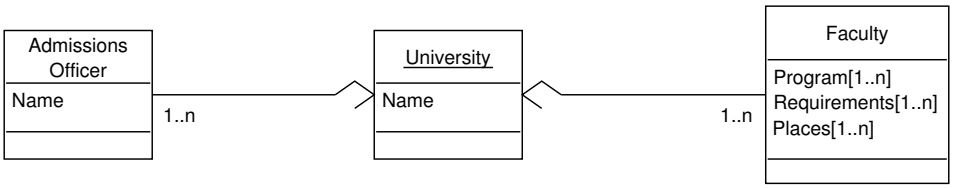


Figure F.11: Faculties

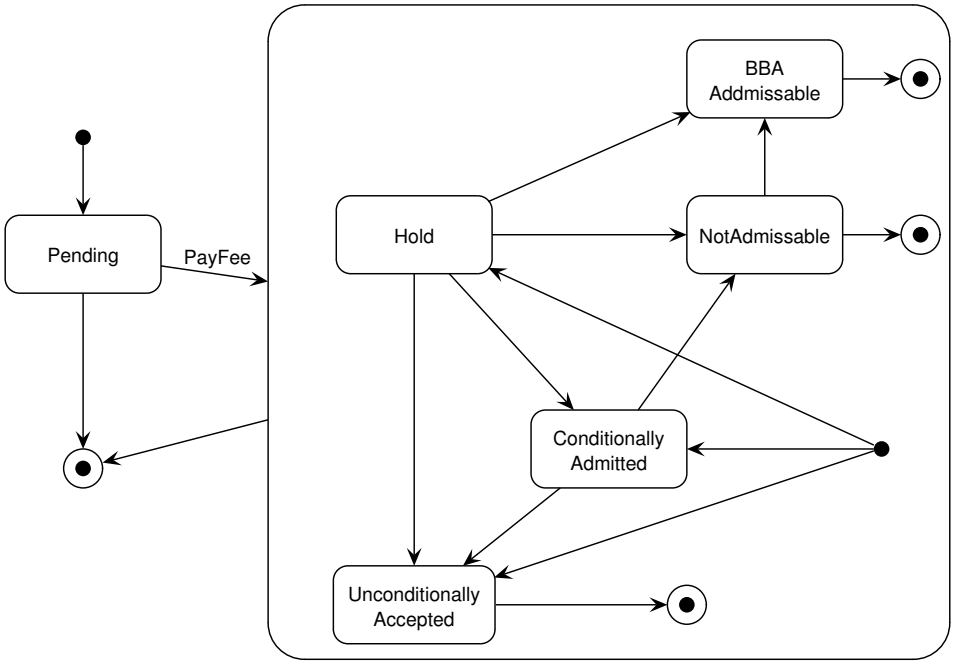


Figure F.12: States of an applicant, incorrect

to change the state, it is the university or parts of the university that take action.

Instead, these states are states of the university, or one of its parts. It is the university (and its parts, the faculties) that possesses all required attributes by rule 21 and it is the admissions officers (parts of the university) that carry out the actions required by rules 25, 26 and corollary 14. Therefore, these states must be modelled as states of the university. Moreover, since the states for each applicant to the university are independent of one another, rule 22 and corollary 16 require that sufficient attributes and association-class attributes are defined. This is here trivially the case, as the university possesses the same independent set of mutual properties with each applicant. Fig. F.13 shows a state chart depicting this interpretation.

Assuming that state chart Fig. F.13 shows (part of) the top-level state chart implies that by rules 25, 26 and corollaries 21, 22 the states are stable and the state transitions between them must be associated with an external event and correspond to an operation of the object/class. In each of the concurrent regions, the second state is a composite state with four sub-states. In the UML meta-model, this corresponds to an embedded state machine, so that these sub-states are not directly part of the top-level state chart. Thus, there does not need to be an external event or an operation of the university (or its parts) associated with the state transitions in that composite state. As a consequence, there exists only one state transition expressing a quantitative change initiated by an external object: 'PayFee' is an event caused by the student object when the FeeBalance, a mutual property, is changed to zero, indicating no outstanding balance. Rules 22 and 15 are satisfied as the two outer states are defined based on the attribute 'FeeBalance' while the sub-states are defined based on the attributes of the association classes representing transcript grades and exam marks.

Other changes to the student and the university are of qualitative nature, when the student applies and becomes an applicant, the university gains a set of state variables which span another orthogonal region in the top-level state chart (rules 22, 32, corollaries 15, 16). Any state transitions that are not associated with external events must be part of a method which realizes an operation which is in turn caused by external events. The method that encompasses the state changes within the sub-state machines is the application procedure that is carried out by the university and its parts, i.e. the faculties and the admissions officer. It is externally induced when the university receives an application from a student accompanied or followed

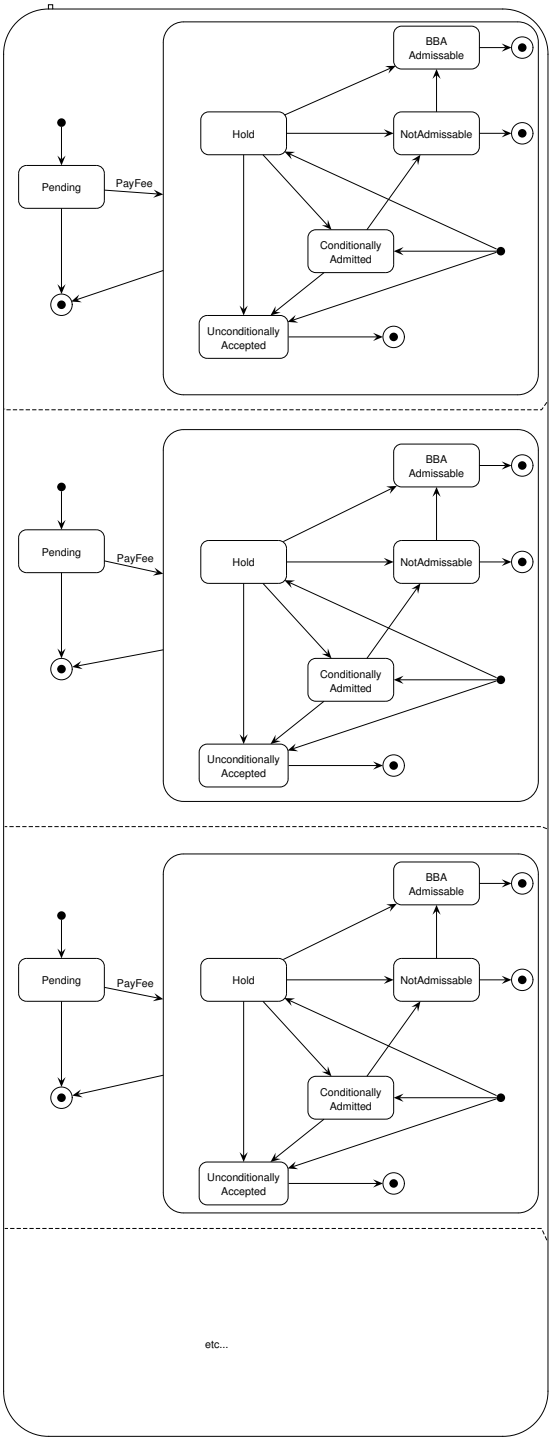


Figure F.13: States of the university

by fee payment.

Once a student is unconditionally accepted, an admissions officer will send an offer of acceptance to the applicant. We again suggest sub-classing such an applicant, for similar reasons that we sub-classed students: Only some of them possess the new properties and only some of them possess new behavioural possibilities (e.g. accepting or rejecting an offer). Once all final transcripts and final provincial ministry grades are received, all applicants on hold and all conditionally accepted students are evaluated and become either unconditionally admissible or must be rejected. In the first case, an admissions officer sends out offers of acceptance, in the second case, a notice of rejection is sent. The amended sequence diagram in Fig. F.14 shows these interactions and Fig. F.15 shows the sub-classification of applicants. Note that rejected applicants are not subclassed, as they possess neither additional properties nor additional behaviour.

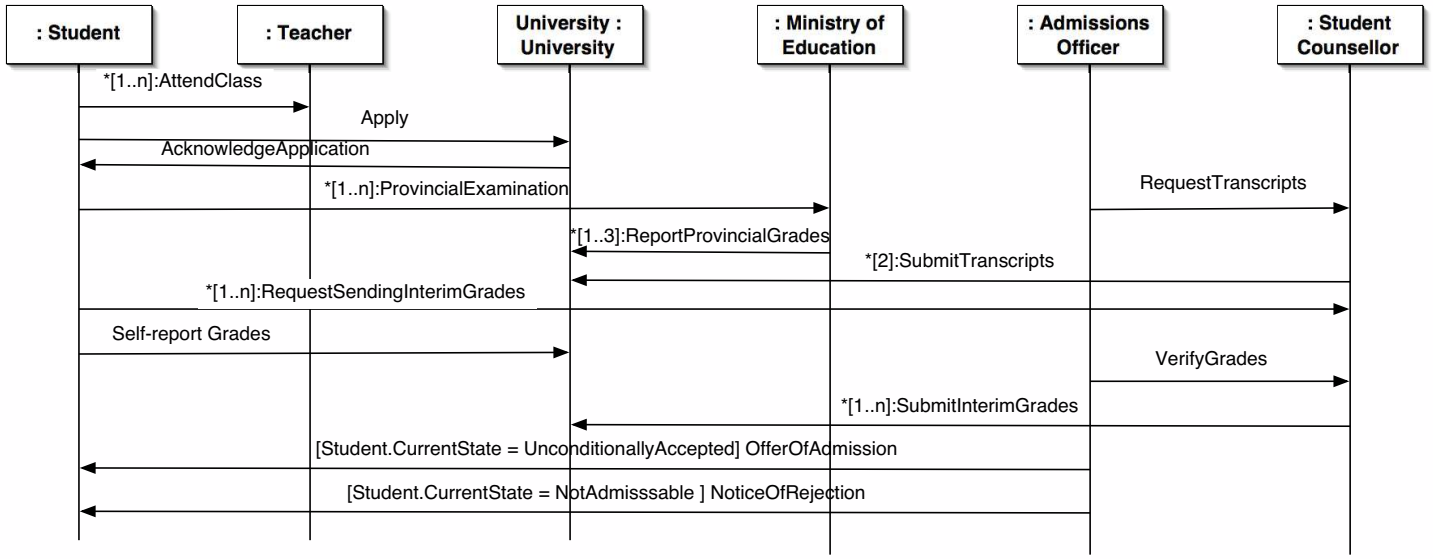


Figure F.14: Acceptance and rejection notices

If the applicant cannot be unconditionally accepted, but the faculty and program applied to has broader based admission (BBA) requirements, the application is referred to faculty admissions officers for evaluations who will then pass a recommendation to a university admission officer who will in turn notify the applicant. BBA requirements are optional properties for faculties. However, if a faculty possesses BBA requirements, it must also have faculty admissions officers. This suggests a subclassification of faculties with such properties and parts, as shown in Fig. F.16.

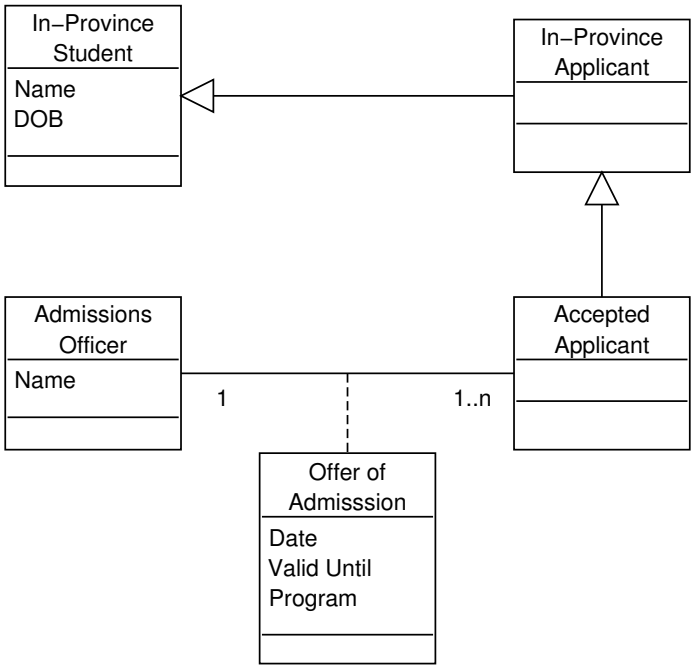


Figure F.15: Accepted and rejected applicants

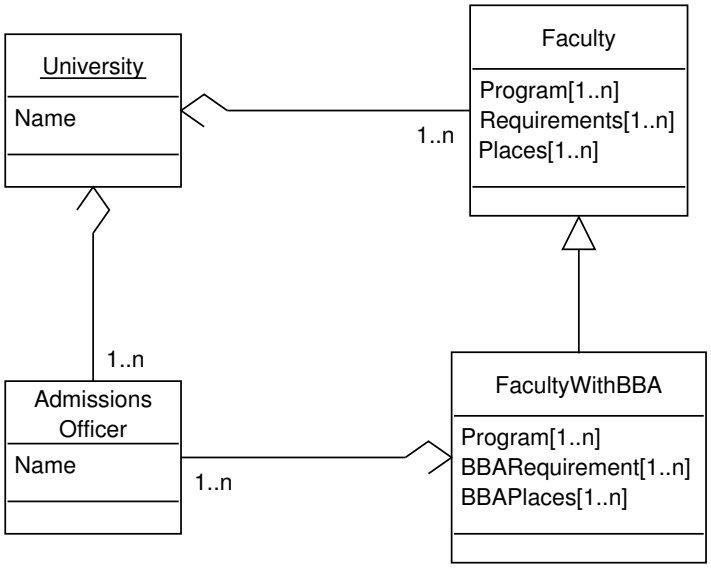


Figure F.16: Faculties with BBA Requirements

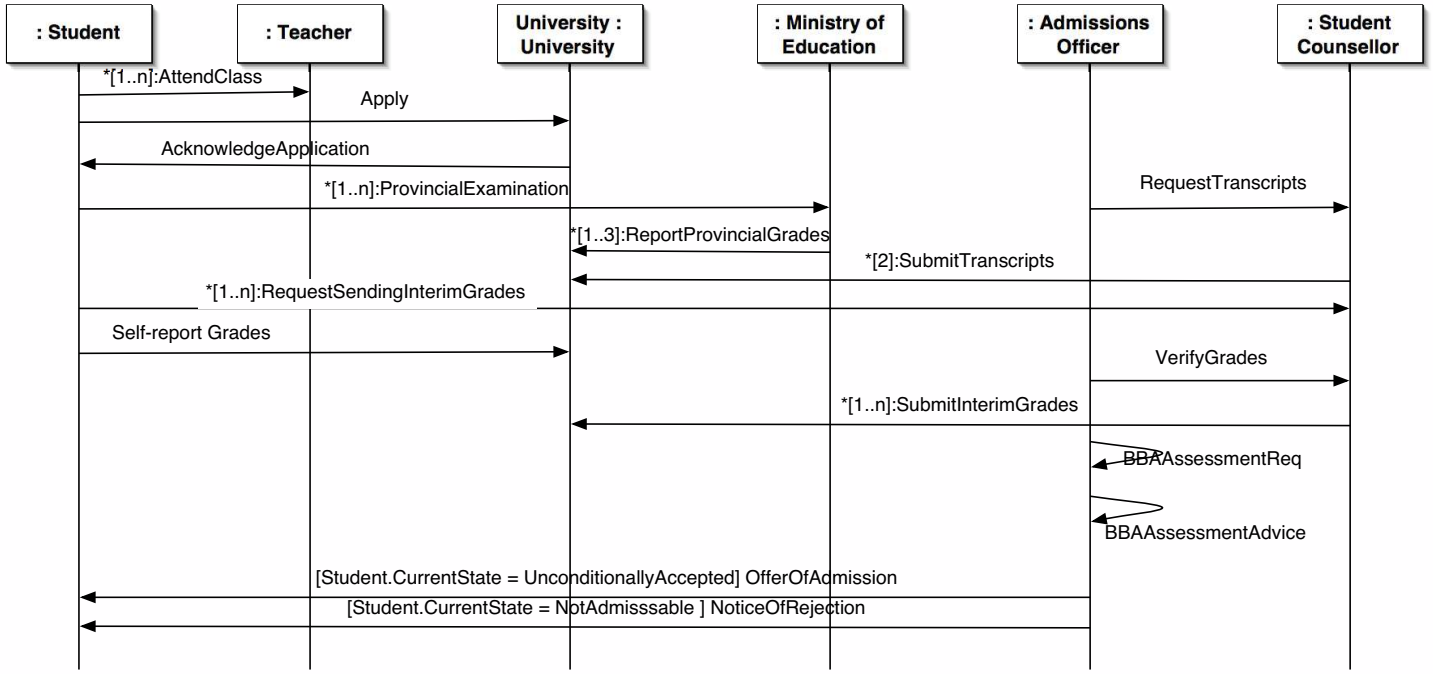


Figure F.17: Faculties with BBA requirements, sequence diagram

The corresponding interactions are shown in Fig. F.17. An admissions officer issues a request to another admissions officer (part of a faculty) and this officer issues a BBA assessment advice back to the first admissions officer.

We next turn to applicants which do not follow the standard process modelled above. The university has agreements with the Ministries of Education of the provinces Ontario and Alberta so that students from these provinces may be treated as in-province students for the purpose of application process. Another group of students are those of the US. These students require an SAT examination in order to gain admission. This is achieved through an interaction with the company administering the SAT tests, ETS, which will electronically transfer the scores to all universities requested by the examinee. This is depicted in Figs. F.18 and F.19.

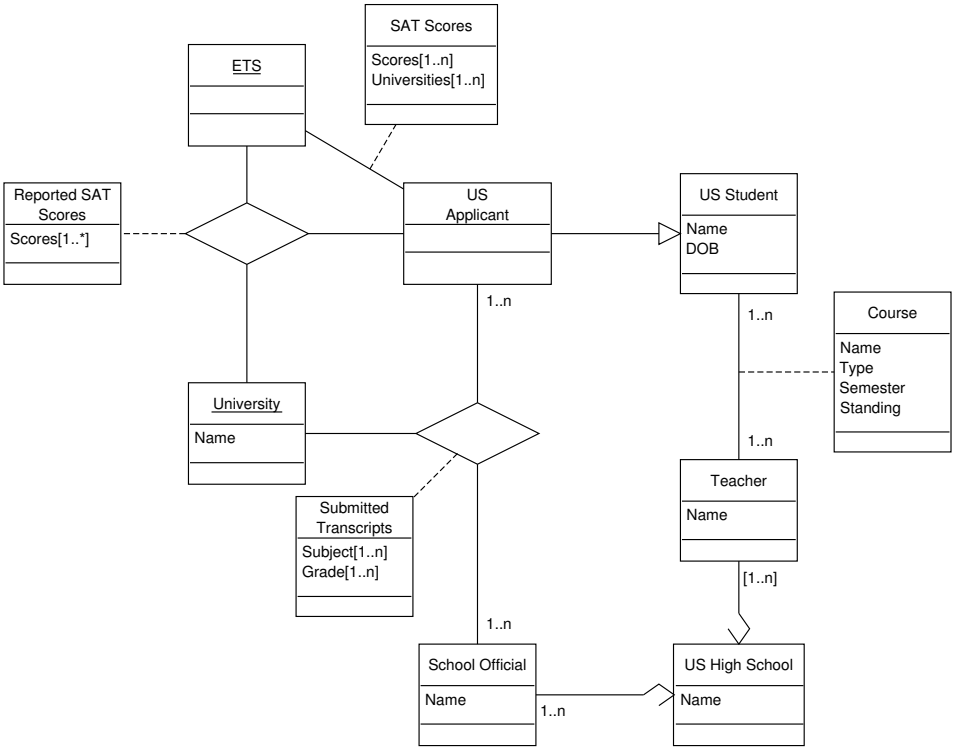


Figure F.18: SAT scores and US applicants

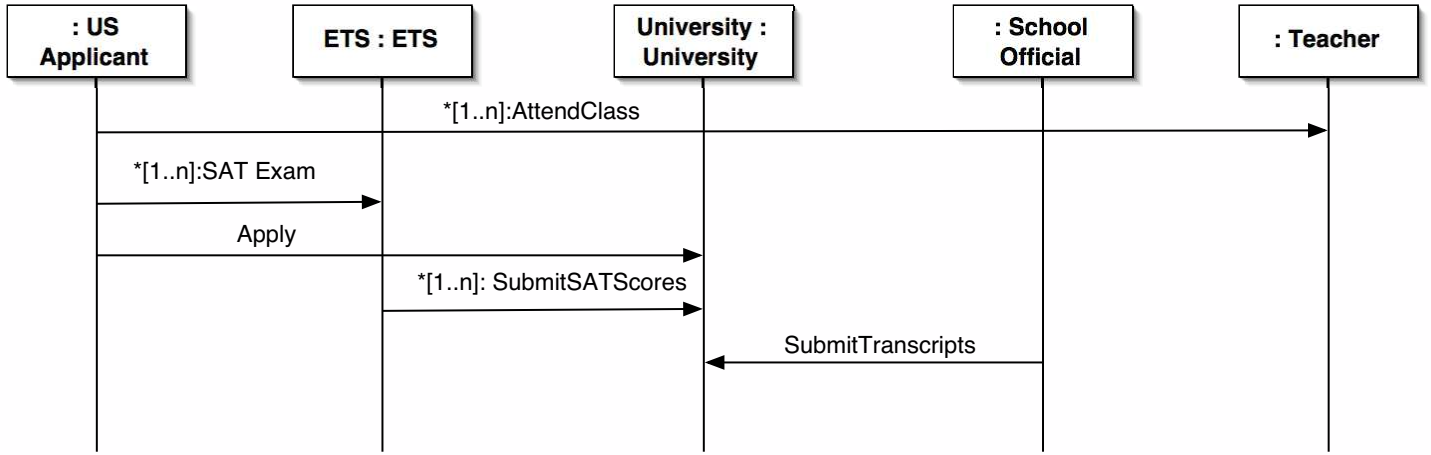


Figure F.19: US applicants interaction

As the university is not able to receive grades in an automated electronic way from the respective ministry of education, the transcripts must be provided directly from the high school, by a high school official. Other aspects of the process and the steps involved in it are identical to the process for local, in-province applicants.

Another group of applicants that is involved in a different set of interactions, resulting in a different set of properties, are international applicants. Such applicants must undergo a test of their English language capabilities with the company administering the standardized TOEFL test (ETS). Results are automatically forwarded to all universities requested by the examinee. They must also submit transcripts translated by a recognized translator or translation service. The interactions are shown in sequence diagram Fig. F.20 and the resulting mutual properties are shown in the class diagram in Fig. F.21.

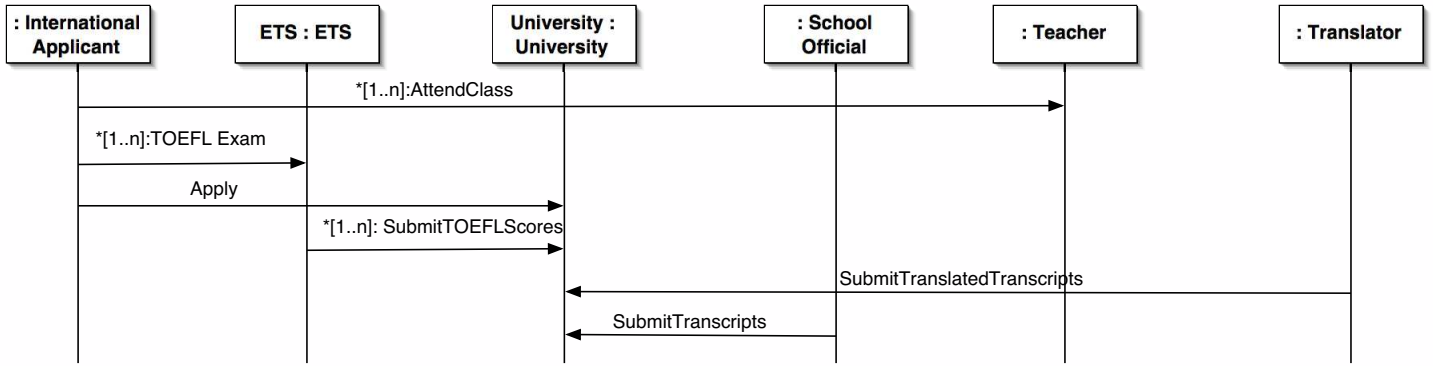


Figure F.20: International applicants interactions

Having defined the three types of applicants separately, we can abstract common features. International students, US students and in-province students are all students which take courses with teachers at high schools. We therefore suggest the generalization structure shown in Fig. F.22. International students and US students are not distinguished based on their properties. Only after they apply, once they are applicants, do they gain different properties, i.e. SAT scores and TOEFL scores etc. As students, all attend high schools and take courses leading to course grades. In-province students on the other hand are different, as they take exams with the Ministry of Education.

There exist two more classes of students, out-of-province students and students in International Baccalaureate (IB) programs. All of their transcripts are submitted by their high school, but they do not require an SAT exam like US applicants do. These applicants form the remainder of the class 'Applicants' which are not instances of the sub-classes 'in-province applicant', 'US applicant' or 'International applicant'. Therefore, no sub-class is necessary to represent these applicants.

Based on this discussion and the generalizations shown in Fig. F.22 we can simplify the previous class diagrams by making use of inheritance (Figs. F.23, F.24, F.25, F.26). The state chart in Fig. F.12 is applicable to all applicants (class 'Applicant').

F.2 Interactions and Operations

The class diagrams developed in the previous paragraphs outline mostly the static structure and represent the things and their (mutual) properties, but neglect to show operations and methods of the objects and classes. Most properties are mutual properties which arise out of and consequently represent interaction. Most of the changes are qualitative changes, expressed through acquisition or loss of properties. As an example, a student becomes an international applicant who becomes an accepted applicant. Only one class, the university, has states defined for it (Fig. F.12). These are based on the values of intrinsic and mutual properties (represented as attributes) and changes of state represent changes to the values of attributes. These quantitative changes and all qualitative changes must be expressed by operations of attributes in the object-oriented paradigm (rules 26, 27). Rule 30 suggests that externally caused events cause an operation to be performed.

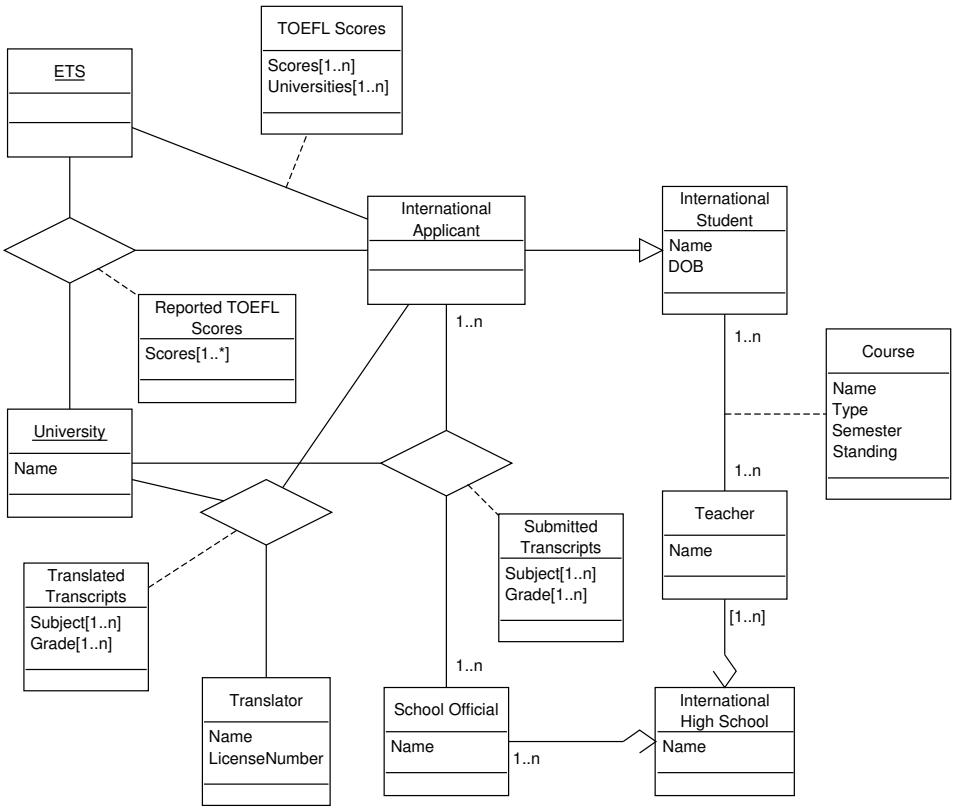


Figure F.21: International applicants

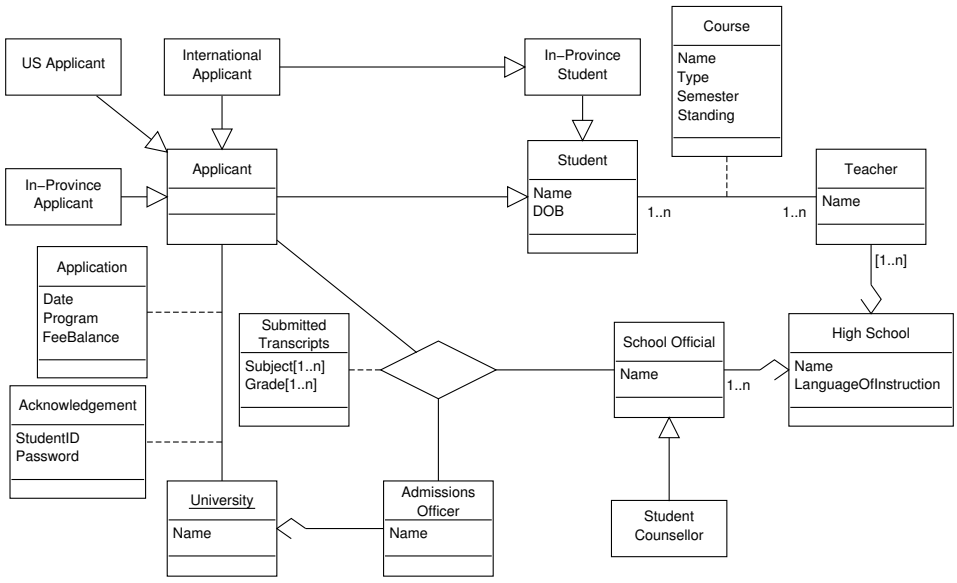


Figure F.22: Generalization of applicants

We can therefore define operations based on external interaction.

A general guidelines that is applicable is based on corollary 5 and rules 4 and 5 require that operations are not modelled on association classes but instead on the participating classes representing the interacting objects.

Some operations and interactions are outside the scope of this analysis, such as the interactions within high schools that lead to the set of properties labelled 'Course'.

As there is very little quantitative change in the analysed domain, rules 25 and 26, stating that the quantitative changes are expressible in state charts and that the transitions of a state chart correspond to operation are not applicable. Moreover, Chapter 6 deals with interactions due to *quantitative change* only and is therefore to a large degree not applicable. Instead, section 4.2.4 provides us with rules 13 and 15, which, together with corollary 11, govern the creation and destruction of objects due to re-classification based on loss or acquisition of (mutual) properties.

The initial interaction is that between a student and the university: The student applies to the university. Once the university has acknowledged the

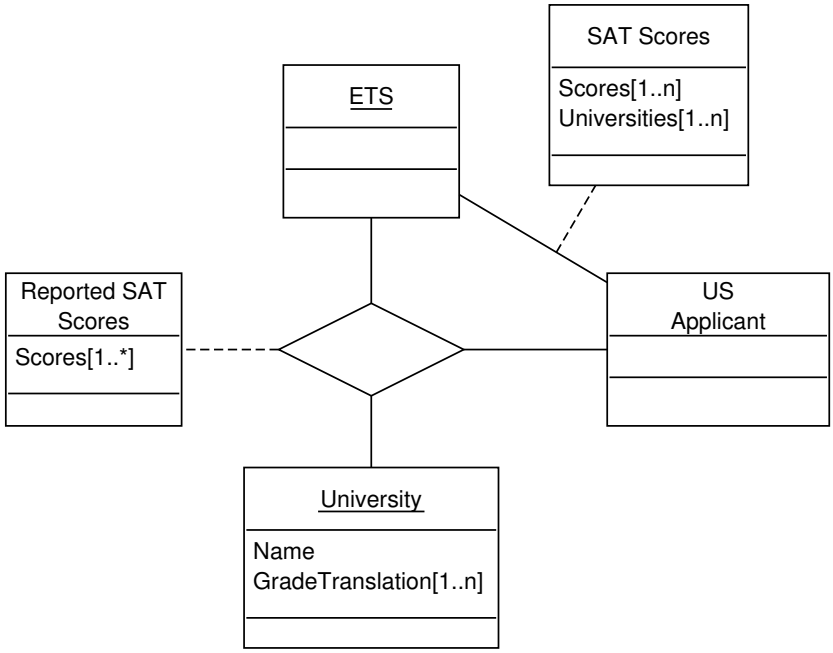


Figure F.23: US Applicants

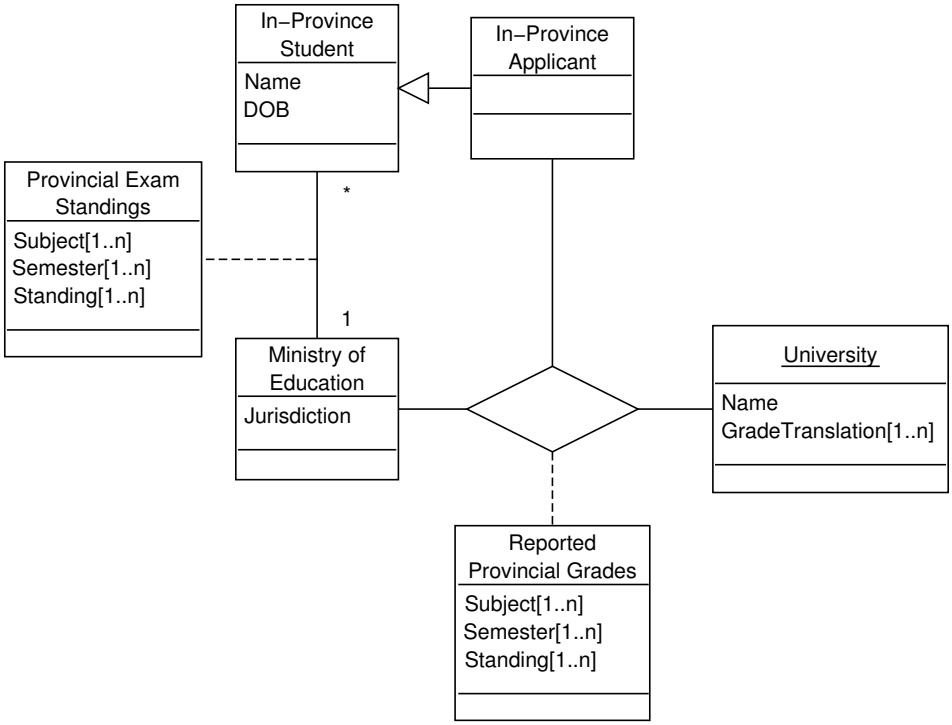


Figure F.24: In-Province Applicants

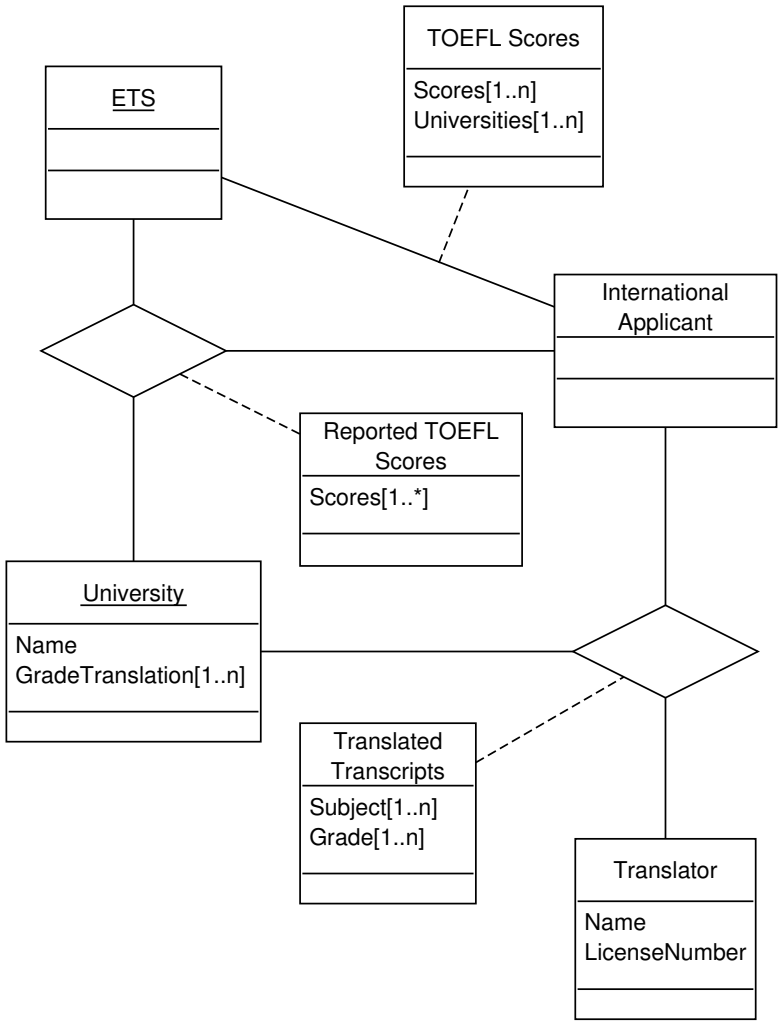


Figure F.25: International Applicants

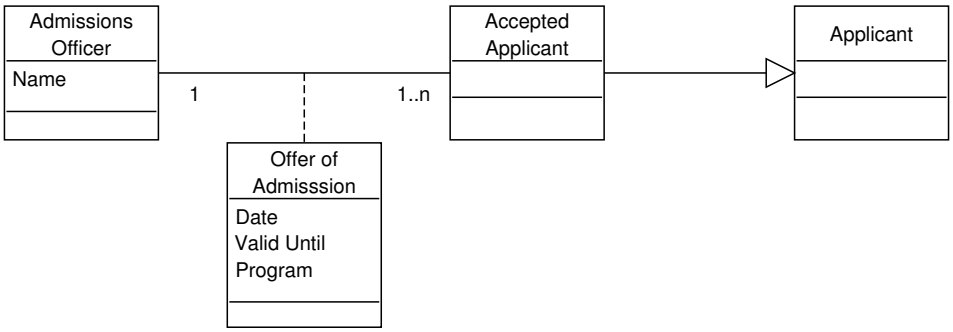


Figure F.26: Accepted Applicants

application, the student becomes an applicant. Thus, an instance of class student needs to be destroyed and an instance of class applicant created with the requisite mutual properties. The interaction 'apply' is not caused by an external event to the student within the scope of this analysis, there must not be an operation. However, the university as receiver of the stimulus must possess an operation which is caused by the event that is caused by reception of the stimulus. We call this operation 'Process Application'. As the changes are of qualitative nature, we do not provide a state chart for a method implementing this method.

As part of the processing of an application, the university sends a message to the student with a student ID and password for services. For the student, this is an external event, and we provide an operation for it. As the student becomes an applicant once all required mutual properties are established, we call the operation 'BecomeApplicant'. Any method implementing this operation must create an instance of applicants and destroy an instance of student as per rules 15 and 13. Students also initiate, as part of their school attendance, interaction with the ministry of interaction. For the ministry, this is an external event requiring an operation. We call this operation 'AdministerExam'. These operations are shown in the class diagram of Fig. F.27.

As part of an internal or ongoing process, the Ministry provides the grades to the university (Figs. F.7, F.9). This therefore does not require an operation for the MoE, but instead must be handled by the university in operation 'AcceptMinistryGrades' (Fig. F.28).

The university, as part of the operation 'ProcessApplication' will also

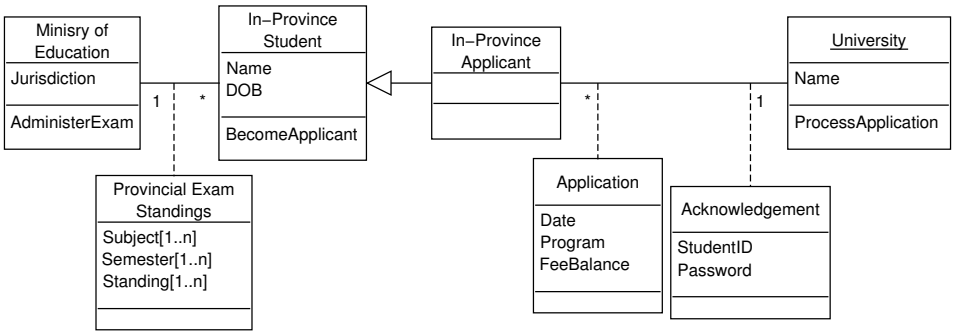


Figure F.27: Operations of students, the university and the MoE

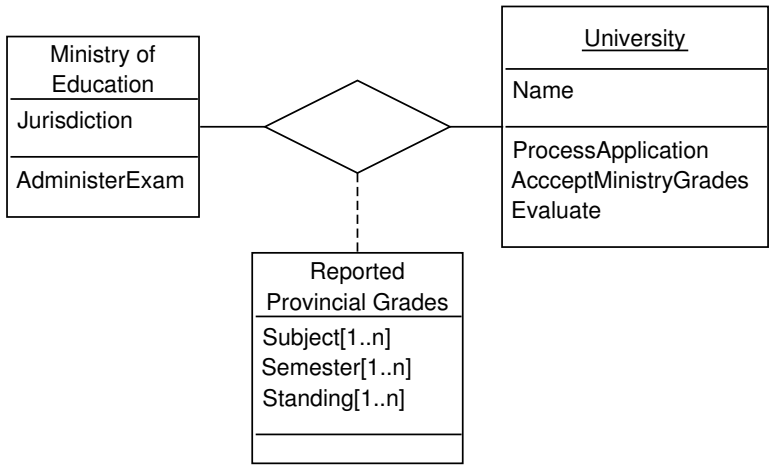


Figure F.28: Operations of the university

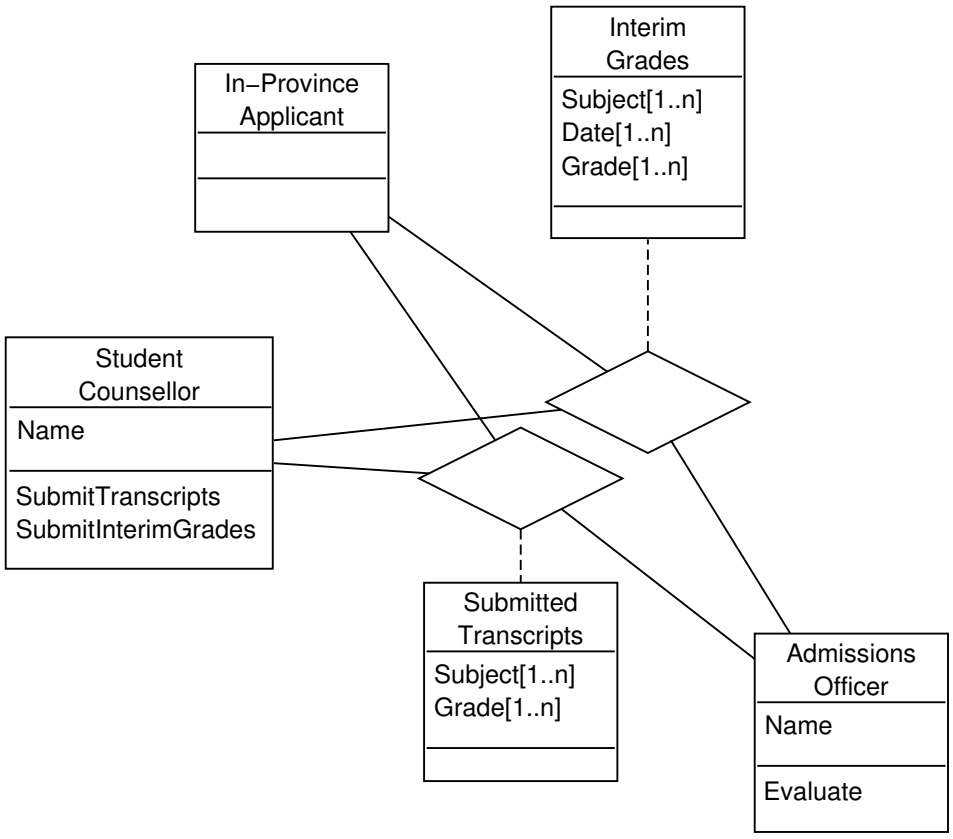


Figure F.29: Operations of student counsellors and admission officers

send requests for transcripts to the high school officials. For the latter, this is an external event and must be processed by an operation, 'SupplyTranscripts'. Student counsellors may also be asked to provide interim grades, requiring operation 'SupplyInterimGrades' (Fig. F.29). Finally, the applicant and student counsellors can report grades or transcripts to the university or the admissions officers. In both cases, internal activity is caused, that of evaluating or re-evaluation an applicant with respect to the requirements. We call these operations of the university and the admissions officers 'evaluate' (Figs. F.28, F.29).

Once the university assesses the admissibility of an applicant and decides whether to issue an offer or rejection notice, it sends one such message to the applicant. This is part of the operation of processing an application or

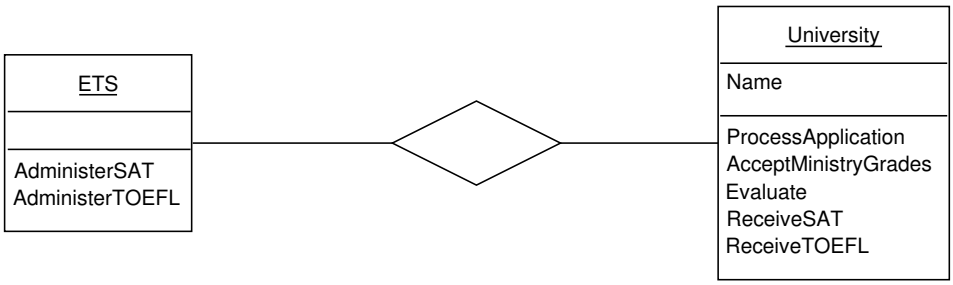


Figure F.30: Operations of the university and ETS

evaluating an applicant, it is therefore not necessary to provide a method for this. For the applicant, this is an initial external event leading to a number of internal changes, such as accepting or rejecting an offer, etc. However, this is outside the scope of the application process under study here, so will not be modelled.

Internal communication between different parts of the university, e.g. admissions officers of the university and admissions officers of the faculties for BBA admissions (see Fig. F.17), are captured by the already defined methods. An admissions officer requesting a BBA evaluation from a faculty admissions officer triggers the operation 'evaluate'. Conversely, once a faculty admissions officer has performed the evaluation, that officer will request an evaluation and notification of the student from the university admissions officer, again triggering the operation 'evaluate'.

The class diagram in Fig. F.30 shows the properties required for the university and ETS to account for external requests from US and international applicants (see sequence diagrams in Figs. F.19 and F.20). Applicants, by taking a SAT or TOEFL exam, cause an external event in the ETS object, which eventually leads to the sending of a message to the university, causing an external event in that object. This causes the behaviour specified in operation 'ReceiveSATScores' and 'ReceiveTOEFLScores'.

Appendix G

Post-Test Questionnaire

G.1 UML Knowledge

UML-A	On a scale of 1 (low) to 7 (high) how do you rate your knowledge of UML?
UML-B	For how many months have you used UML in practice (outside classes or courses)?
UML-1	An aggregation expresses a *Part-Of* relationship between a component object and an aggregate object. (True/False)
UML-2	A subclasss inherits all the features from its superclass. (True/False)
UML-3	On a class diagram, a ternary relationship is represented by a rounded rectangle. (True/False)
UML-4	Abstracting the common features among multiple classes, as well as the relationships they participate in, is called generalization. (True/False)

UML-5 An object class is a set of objects that share a common structure and a common behavior. (True/False)

UML-6 On a class diagram, an exclamation point represents a multiplicity with an infinite upper bound. (True/False)

UML-7 On a class diagram, an asocation is signified by a douple-ended arrow that connects the participating object classes. (True/False)

UML-8 An object is an entity that has a well-defined role in the application domain, and has state, behaviour, and identity. (True/False)

UML-9 When indicating the multiplicity for a role, an infinite upper bound is denoted by a

1. Dash
2. Diamond
3. Hollow point arrow
4. Star

UML-10 In UML, a class is represented by

1. A rectangle with three compartments separated by horizontal lines.
2. A circle in which the activity name is recorded.
3. A double-lined ellipse in which the activity name is recorded.
4. A diamond in which the activity is recorded.

UML-11

Changes in the attributes of an object or in the links an object has with other objects best defines:

1. Events
2. Operation
3. State Transition
4. Method

UML-12

A function or a service that is provided by all the instances of class best defines

1. Encapsulation
2. Task Set
3. Operation
4. Multiplicity

UML-13

A relationship among instances of object classes best defines

1. Encapsulation
2. Scope
3. Association
4. Composition

UML-14

Which of the following indicates how many objects participate in a given relationship?

1. Assocation role
2. Object count
3. Multiplicity
4. Association class

UML-15

The degree of an association relationship can be:

1. Unary
2. Binary
3. Ternary
4. All of the above

UML-16

The end of an association where it connects to a class best describes:

1. Encapsulation
2. Scope
3. Association role
4. Composition

UML-17

Which of the following encompasses an object's properties and the values those properties have?

1. Behaviour
2. Class
3. State
4. Encapsulation

UML-18

Which of the following indicates a minimum of 0 and a maximum of 1?

1. 1 . . . 0
2. 0 . . . 1
3. 1 – 0
4. 1:M

UML-19	Which of the following is not a true statement? 1. An object's behaviour depends on its state and the operation being performed. 2. An object's state is determined by its attribute values and links to other objects. 3. An operation is simply an action that one object performs upon another in order to get a response. 4. Object class refers to an entity that has a well-defined role in the application domain, and has state, behaviour and identity.

G.2 Ease of Interpretation, Usefulness and Information Content

EOI-1	The class diagram was easy to read.
EOI-2	The class diagram was clear and understandable.
EOI-3	The class diagram was easy to understand.
EOI-4 (R) (*)	Interpreting the class diagram was frustrating.
EOI-5 (*)	The class diagram was simple.
EOI-6 (R)	I believe that the class diagram was cumbersome to interpret.
EOI-7	I feel comfortable with interpreting the class diagram.
EOI-8	I believe that it is easy to interpret the class diagram.
EOI-9 (R)	Interpreting the class diagram required a lot of mental effort.
EOI-10	Overall, I found the class diagram easy to interpret.
EOI-11	It was easy for me to understand what the diagram was trying to model.
EOI-12 (*)	Overall, I found the class diagram was easy to use.
USE-1 (*)	I feel confident that my answers are correct.
USE-2	The information in the class diagram was helpful for answering the questions.
USE-3	Interpreting the class diagram increased my effectiveness in answering the questions.

USE-4	Interpreting the class diagram improved the quality of my answers to the questions.
USE-5	Understanding the class diagram made answering the questions easier.
USE-6	Interpreting the diagram enhanced the quality of my answers.
INFO-1	I believe the diagram was an accurate representation of order processing[1].
INFO-2	The class diagram provided all the information required to answer the questions.
INFO-3 (*)	Interpreting the class diagram has provided knowledge about order processing[2].
INFO-4	I believe the diagram showed a lot of detail.
INFO-5 (*)	The diagram was a good diagram.

Note 1: EOI : Ease of Interpretation; USE : Usefulness; INFO : Information content

Note 2: (R) indicates reverse coded items.

Note 3: (*) indicates the item was not included in the final instrument.

Note 4: The item names may be prefixed according to whether they refer to the order processing domain or the car rental domain, e.g. CR-EOI-1 refers to the first item on the Ease of Interpretation scale for the car rental domain.

G.3 Domain Familiarity

OP-1	On a scale of 1 (low) to 7 (high) how do you rate your knowledge of order processing procedures?
OP-2	Have you ever worked in an order processing department? (yes/no)

[1] Car Rental, depending on the domain.
[2] Car Rental, depending on the domain.

CR-1	On a scale of 1 (low) to 7 (high) how do you rate your knowledge of car rental procedures?
CR-2	Have you ever rented a car before? (yes/no)
CR-3	Have you ever worked for a car rental agency? (yes/no)

Appendix H

Diagram Comprehension Questions

Order Processing Domain

OPComp-1	Is the corporate customer billed on a monthly basis?
OPComp-2	Do all customers have a sales representative ?
OPComp-3	Does the order show taxes paid on items?
OPComp-4	Can an order contain products of different types?
OPComp-5	Is a sales representative responsible for a product type?
OPComp-6	Can customers pay for their orders by credit card?
OPComp-7	Does the order total show discounts?
OPComp-8	Do corporate customers receive payment reminders?
OPComp-9	Can a customer order the same products multiple times?
OPComp-10	Can orders be prepaid?
OPComp-11	Can a corporate customer have multiple sales representatives?

Car Rental Domain

CRComp-1	Can the pick-up location for cars be different from the drop-off location?

CRComp-2	Can customers reserve multiple cars?
CRComp-3	Are the license plate numbers of rented vehicles recorded?
CRComp-4	Can a reservation be billed multiple times?
CRComp-5	Is there a minimum age for customers to rent a vehicle?
CRComp-6	Can customers purchase insurance for the vehicle they rent?
CRComp-7	Does the company charge freight and inspection for sold cars?
CRComp-8	Does the invoice show taxes?
CRComp-9	Can an invoice be produced without renting a car?

Appendix I

Problem Solving Questions

Order Processing Domain

OPProb-1	Suppose that an important customer needs to order products urgently? What problems could he/she face?
OPProb-2	Suppose that a shipment does not contain all ordered products. What could have happened?
OPProb-3	Suppose the warehouse has run out of items and an order cannot be fulfilled. How could this have happened?
OPProb-4	Suppose two customer's orders are mixed up. What could have happened?
OPProb-5	Suppose that a customer wants to change their order. What problems could this pose?
OPProb-6	Suppose the customer wannts to know the expected delivery date of his order? What problems might be experienced when replying to the customer?
OPProb-7	Suppose a customer is not billed for some products. What could have happened?

Car Rental Domain

CRProb-1	Suppose that a customer arrives for pick-up but no car is available. What could have happened?

CRProb-2	Suppose a customer receives two invoices. What could have happened?
CRProb-3	Suppose a car which is reserved for a customer is being sold at auction. How could this have happened?
CRProb-4	Suppose a customer is not billed for a reservation? What could have happened?
CRProb-5	Suppose that a customer wants to extend his rental period. What could go wrong?

Appendix J

Experimental Results (Extended Analysis)

J.1 ANCOVA (Step-wise Inclusion of Variables)

In order to arrive at a more parsimonious explanatory model than the full ANCOVA model presented in Section 11.8.4, non-significant factors and covariates were omitted from the model and interaction effects of UML knowledge were considered. This supplementary analysis reveals that only the variables "Rules", "Group" and "UML.TTL" need to be considered.

```
Error: Subject
      Df Sum Sq Mean Sq
Rules  1 1.3804  1.3804

Error: Within
            Df Sum Sq Mean Sq F value    Pr(>F)
Rules        2  3.574   1.787  4.0133 0.0212222 *
Group        2  5.889   2.944  6.6126 0.0020477 **
UML.TTL      1  5.002   5.002 11.2348 0.0011535 **
Rules:Group  4  9.324   2.331  5.2352 0.0007447 ***
Residuals   95 42.299   0.445
---
Signif. codes:  0 ***' 0.001 **' 0.01 *' 0.05 .' 0.1  ' 1
```

Overall, the fit of this reduced model is $R^2 = .3731$ which is only marginally smaller than the goodness of fit of the complete model, suggesting that this more parsimonious model offers as good an explanation for the observed data.

To further assess the results and confirm the more parsimonious model, it was decided to successively include factors and covariates in the model while the model quality increases. The statistical package `R` provides a procedure for doing this, based on the Akaike Information Criterion (AIC) which assesses model fit and is a function of the number of model parameters to be fitted so as to avoid overfitting. With this interpretation, a smaller AIC is better (Pinheiro and Bates, 2000). Factors and covariates are included as long as the AIC of the resulting model can be lowered.

```
Start:  AIC= -45.89
 Prob ~ 1

                Df Sum of Sq     RSS     AIC
+ UML.TTL        1     7.215  60.253 -55.878
+ Group          2     6.544  60.924 -52.704
+ Rules          2     4.112  63.356 -48.555
+ UML.A          1     2.517  64.952 -47.919
+ Information    1     2.417  65.051 -47.756
<none>                        67.468 -45.889
+ Time           1     0.952  66.516 -45.396
+ Usefulness     1     0.730  66.739 -45.041
+ SelfAssess     1     0.265  67.203 -44.306
+ Domain         1     0.195  67.274 -44.195
+ Comp           1     0.095  67.373 -44.038
+ Interpretation  1 5.326e-05  67.468 -43.889

Step:  AIC= -55.88
 Prob ~ UML.TTL

                Df Sum of Sq     RSS     AIC
+ Group          2     4.386  55.868 -59.888
+ Rules          2     3.556  56.697 -58.327
+ Time           1     1.342  58.911 -56.265
<none>                        60.253 -55.878
+ Information    1     1.066  59.187 -55.770
```

```
+ Usefulness      1     0.615  59.638 -54.966
+ SelfAssess      1     0.383  59.870 -54.554
+ UML.A           1     0.318  59.935 -54.439
+ Domain          1     0.195  60.058 -54.221
+ Comp            1     0.036  60.217 -53.941
+ Interpretation  1     0.001  60.252 -53.879
- UML.TTL         1     7.215  67.468 -45.889

Step:  AIC= -59.89
 Prob ~ UML.TTL + Group

                 Df Sum of Sq     RSS     AIC
+ Rules           2     4.071  51.797 -63.907
<none>                        55.868 -59.888
+ SelfAssess      1     0.709  55.159 -59.242
+ Usefulness      1     0.570  55.297 -58.976
+ Information     1     0.530  55.337 -58.899
+ Time            1     0.455  55.413 -58.754
+ Domain          1     0.195  55.673 -58.258
+ UML.A           1     0.147  55.721 -58.167
+ Interpretation  1     0.070  55.798 -58.021
+ Comp            1     0.008  55.859 -57.904
- Group           2     4.386  60.253 -55.878
- UML.TTL         1     5.056  60.924 -52.704

Step:  AIC= -63.91
 Prob ~ UML.TTL + Group + Rules

                 Df Sum of Sq     RSS     AIC
<none>                        51.797 -63.907
+ SelfAssess      1     0.949  50.848 -63.868
+ Comp            1     0.776  51.021 -63.507
+ Time            1     0.663  51.134 -63.273
+ Information     1     0.519  51.278 -62.975
+ Usefulness      1     0.229  51.568 -62.378
+ Domain          1     0.195  51.602 -62.307
+ Interpretation  1     0.063  51.734 -62.036
+ UML.A           1     0.002  51.795 -61.912
- Rules           2     4.071  55.868 -59.888
- Group           2     4.900  56.697 -58.327
```

```
- UML.TTL         1     4.829  56.626 -56.460

Call:
lm(formula = Prob ~ UML.TTL + Group + Rules, data = gData)

Coefficients:
(Intercept)      UML.TTL       GroupC      GroupC2
    0.01521      0.11382     -0.46989     -0.07550

     RulesR      RulesR2
    0.32309     -0.15017
```

This procedure produces a model with the same factors and covariates as the reduced model assessed with ANCOVA in Sec. 11.8.4. The algorithm adds that element to the model which increases the model quality (decreases the AIC) most. Thus, in the first step, "UML.TTL" is included, followed by "Group" and "Rules". No interaction effects are considered.

J.2 Linear Mixed Effects Modelling (Analysis of Results)

Linear mixed effect (LME) models (Pinheiro and Bates, 2000) work by (restricted) maximum likelihood fitting of parameters to the data and allow modelling of nested random and nested fixed effects within the same model. Similar to the ANCOVA methodology employed in Sec. 11.8.4, the analysis begins with a complete model including all factors and covariates, including second order factor interaction terms. Goodness of the model is again assessed using the AIC criterion discussed above.

```
Linear mixed-effects model fit by REML
 Data: gData
       AIC      BIC    logLik
  271.7827 330.1223 -111.8913

Random effects:
 Formula: ~1 | Subject
        (Intercept)  Residual
StdDev:   0.4966753 0.4682717
```

```
Fixed effects: Prob ~ (Rules + Domain + Group)^2 + UML.TTL
+ UML.A + Time + Information + Interpretation + Usefulness
+ Comp + SelfAssess
                      Value Std.Error DF    t-value p-value
(Intercept)      -1.1069428 1.0469954 42 -1.0572566  0.2964
RulesR            1.1662674 0.4235645 42  2.7534588  0.0087
RulesR2           0.3790730 0.4280563 42  0.8855682  0.3809
DomainOP         -0.2521666 0.2335060 42 -1.0799148  0.2863
GroupC           -0.0355587 0.3822908 42 -0.0930149  0.9263
GroupC2           0.7427533 0.5323224 42  1.3953073  0.1703
UML.TTL           0.0805366 0.0513957 42  1.5669925  0.1246
UML.A             0.0110384 0.1432621 42  0.0770506  0.9389
Time              0.0256997 0.0109856 42  2.3393879  0.0241
Information       0.1183118 0.0662602 42  1.7855623  0.0814
Interpretation   -0.0231100 0.0643543 42 -0.3591066  0.7213
Usefulness        0.0978097 0.0692240 42  1.4129444  0.1650
Comp              0.8690636 0.5968795 42  1.4560119  0.1528
SelfAssess        0.0571151 0.0556054 42  1.0271512  0.3102
RulesR:DomainOP  -0.2128368 0.2530314 42 -0.8411479  0.4050
RulesR2:DomainOP  0.1705046 0.2387212 42  0.7142416  0.4790
RulesR:GroupC    -1.1101774 0.4913256 42 -2.2595555  0.0291
RulesR2:GroupC   -0.6597350 0.5103743 42 -1.2926494  0.2032
RulesR:GroupC2   -0.6972327 0.6766675 42 -1.0303920  0.3087
RulesR2:GroupC2  -1.3747285 0.6325665 42 -2.1732553  0.0355
DomainOP:GroupC   0.3668574 0.2344418 42  1.5648118  0.1251
DomainOP:GroupC2 -0.2240988 0.2677295 42 -0.8370344  0.4073

Standardized Within-Group Residuals:
        Min          Q1         Med          Q3         Max
-1.73759604 -0.50887871 -0.04269248  0.42715819  1.93127490

Number of Observations: 106
Number of Groups: 53
```

The results above indicate similar effects as shown in the ANCOVA results. The above table shows the estimated value of the effect size and the t-statistic and p-value for significance. The t-statistics and t-test shown is a test for marginal significance *of a particular estimated parameter, not a model term*. It shows whether this particular model parameter is sig-

nificantly different than the baseline of the factor. The results show that diagrams conforming to rules ("R") have a significantly (p=.0087) different effect on problem solving performance than the other two types of diagrams ("N" and "R2"). Also, the subjects in group "C" interpreting diagrams of type "R" behave differently (p=.0291) than other combinations of subject groups and diagram types. Pinheiro and Bates (2000) suggest that an F-test for inclusion of model elements provides a better criterion:

```
                numDF denDF  F-value p-value
(Intercept)         1    42 361.5242  <.0001
Rules               2    42   2.8853  0.0670
Domain              1    42   0.8879  0.3514
Group               2    42   4.7220  0.0141
UML.TTL             1    42   6.7755  0.0127
UML.A               1    42   0.0032  0.9549
Time                1    42   2.1374  0.1512
Information         1    42   3.0173  0.0897
Interpretation      1    42   0.0335  0.8556
Usefulness          1    42   2.2705  0.1393
Comp                1    42   0.5162  0.4764
SelfAssess          1    42   0.6216  0.4349
Rules:Domain        2    42   0.6873  0.5085
Rules:Group         4    42   2.8122  0.0372
Domain:Group        2    42   3.2396  0.0492
```

These results indicate that the factor "Rules" tends towards significance while confirming that "Group" and "UML.TTL" are significant model elements. Similarly, the domain is not significant but the interaction of Rules and Group is. These results are in accord with those obtained by the full ANCOVA model in Sec. 11.8.4.

In order to find the most parsimonious model to explain the data, one can follow a similar procedure as in the ANCOVA based analysis in Appendix J.1. Model elements can be included and the resulting change in goodness of fit as per the AIC criterion is assessed. Following this procedure leads to the same reduced model that was arrived at in the ANCOVA case.

J.3 Verification of Assumptions

Pinheiro and Bates (2000) point out two main assumptions for LME models: (1) The within group errors (here: within subject errors) are IID[1] with zero mean and are homoscedastic[2]. They must also be independent of the random effects. (2) The random effects are normally distributed and independent for different groups. These two assumptions must also hold for an ANOVA procedure to produce valid results. The following paragraphs follow the procedure outlined in (Pinheiro and Bates, 2000).

The first assumption can be assessed by examining the within-subject residuals as they provide surrogates for the error terms. The boxplot in Fig. J.1 shows that the residuals are independent for different subjects and appear to be centered around a mean of zero. Figures J.2 through J.4 plot the residuals against the fitted values for different factor levels of "Group", "Domain", and "Rules" respectively.

None of these plots indicate different residuals within different subsamples. Heteroscedasticity is also formally evaluated by comparing different models. The base model analyzed above is compared against models which allow for heteroscedasticity between different sample subgroups:

```
> fitted1a <-
    update (fitted1, weights=varIdent(form = ~ 1 | Group))
> anova(fitted1, fitted1a)
         Model df      AIC    logLik   Test L.Ratio p-value
fitted1      1 24 271.7827 -111.8914
fitted1a     2 26 270.7511 -109.3756 1 vs 2 5.03157  0.0808
> #
> fitted1b <-
    update (fitted1, weights=varIdent(form = ~ 1 | Domain))
> anova(fitted1, fitted1b)
         Model df      AIC    logLik   Test   L.Ratio p-value
fitted1      1 24 271.7827 -111.8914
fitted1b     2 25 273.4269 -111.7135 1 vs 2 0.3557705  0.5509
> #
> fitted1c <-
    update (fitted1, weights=varIdent(form = ~ 1 | Rules))
```

[1] Independent and identically distributed.
[2] Equal variance

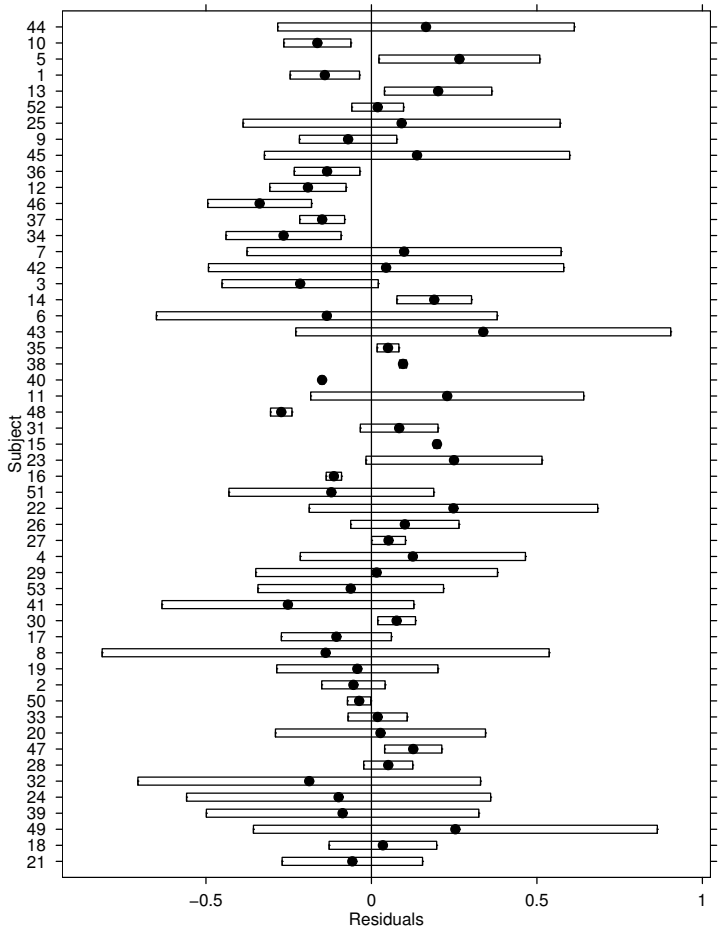


Figure J.1: Box plot of residuals of LME fit

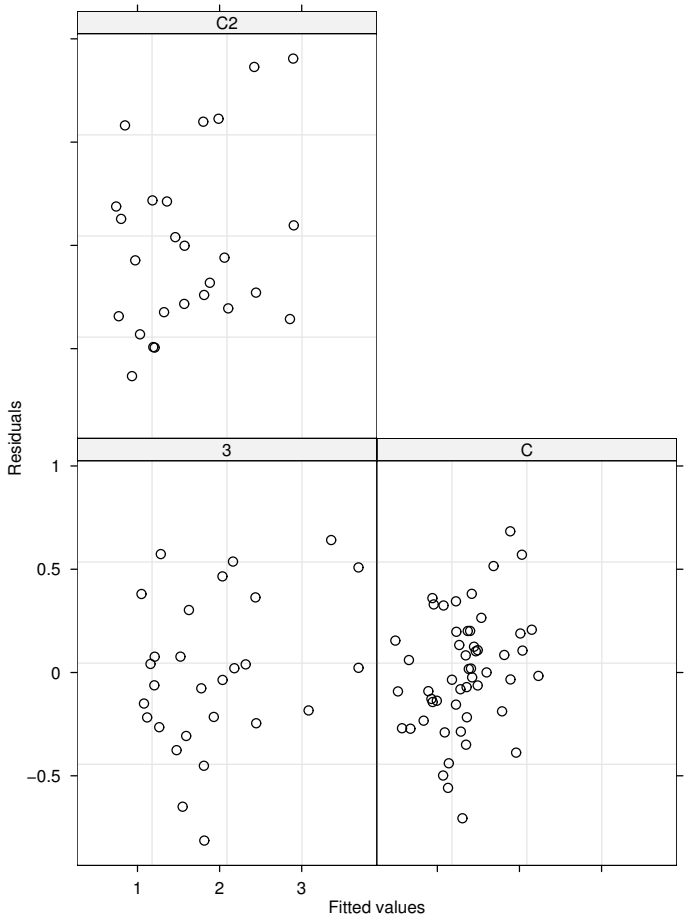


Figure J.2: Residuals against fitted values by group

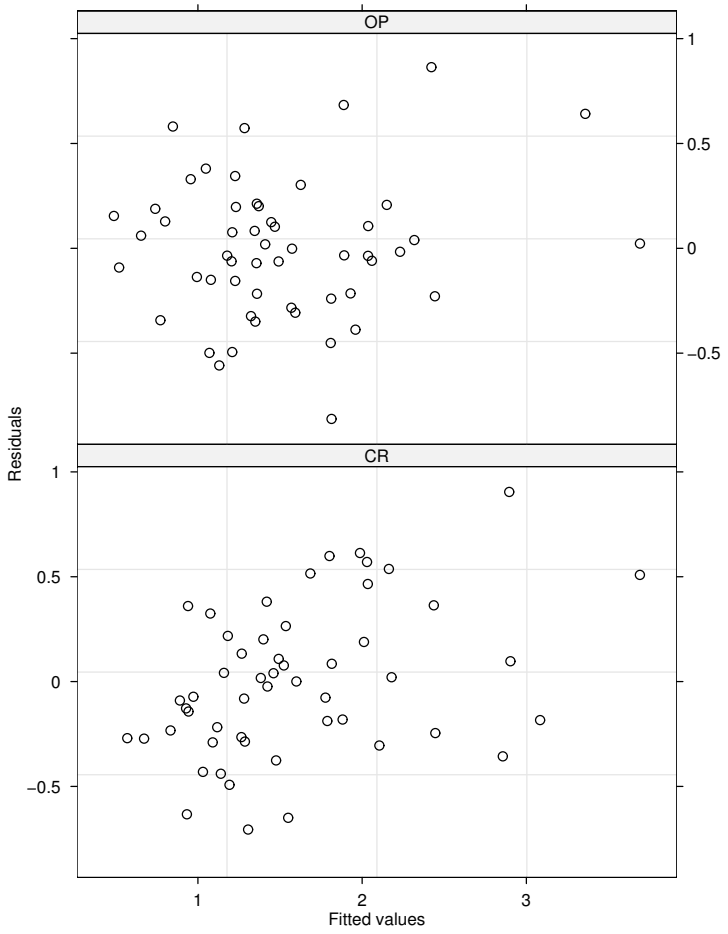


Figure J.3: Residuals against fitted values by domain

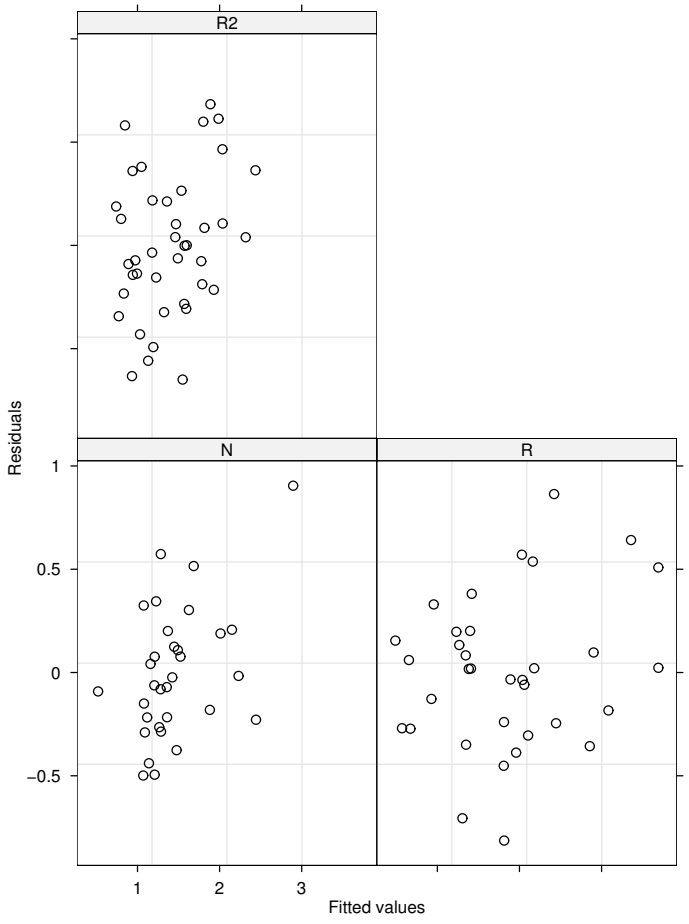


Figure J.4: Residuals against fitted values by type of diagram

```
> anova(fitted1, fitted1c)
         Model df      AIC    logLik   Test   L.Ratio p-value
fitted1      1 24 271.7827 -111.8914
fitted1c     2 26 275.0048 -111.5024 1 vs 2 0.7778486  0.6778
```

None of the differences are statistically significant. While there appear to be indications for different variances between the groups of subjects (p=.0808), the differences in model quality as measured by the AIC criterion do not suggest any practical significance of such differences. Overall, the residuals appear small which indicates a good fit of the fitted with the observed values as shown in Fig. J.5.

Normality of the residuals is assessed by plotting them against the quantiles of the normal standard distribution, as shown in Figures J.6 through J.9, for the total sample and again for sub-samples by factor levels. The resulting plots show that the normality assumption is justified, indicated by the relatively straight and symmetric pattern and clustering towards the center.

A similar plot can be done for the random effects of each subject, shown in Fig. J.10. This shows that the normality assumption for random effects is justified.

J.4 Equivalence of Diagrams

One of the main assumptions was that our models should be informationally equivalent with respect to information relevant to the task. This is assessed by analyzing the effect of the different model types on the factors Information Content, Usability and Ease of Interpretation. This is done using a standard ANOVA procedure including "Domain", "Group" and "Rules" together with second order interactions as independent variables (Appendix K.8).

- Information Content

```
Error: Subject
      Df  Sum Sq Mean Sq
Rules  1 0.85165 0.85165

Error: Within
```

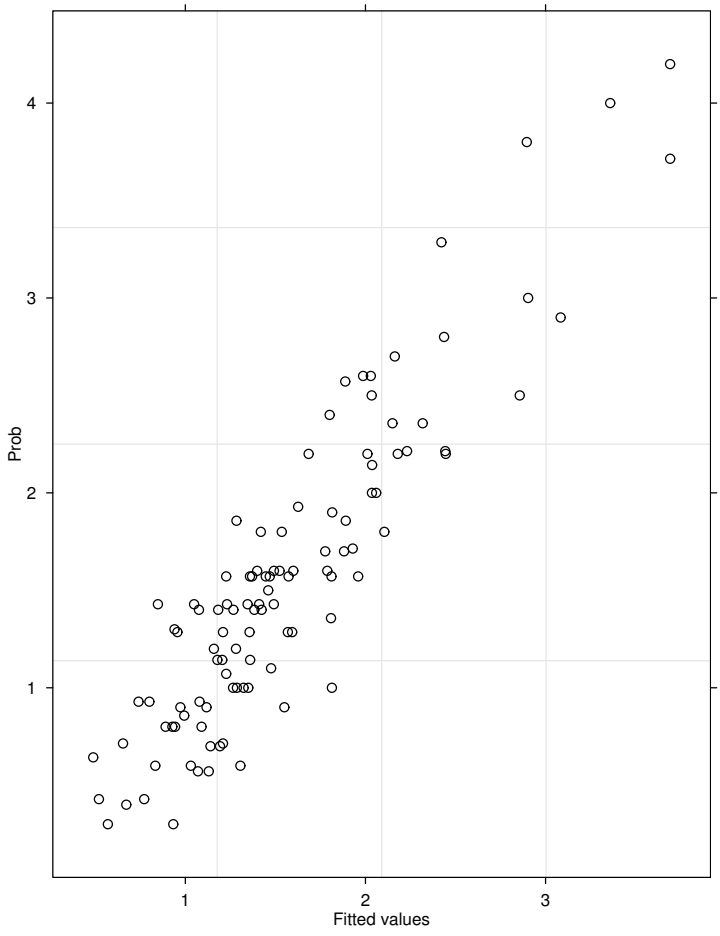


Figure J.5: Fitted values against observed values

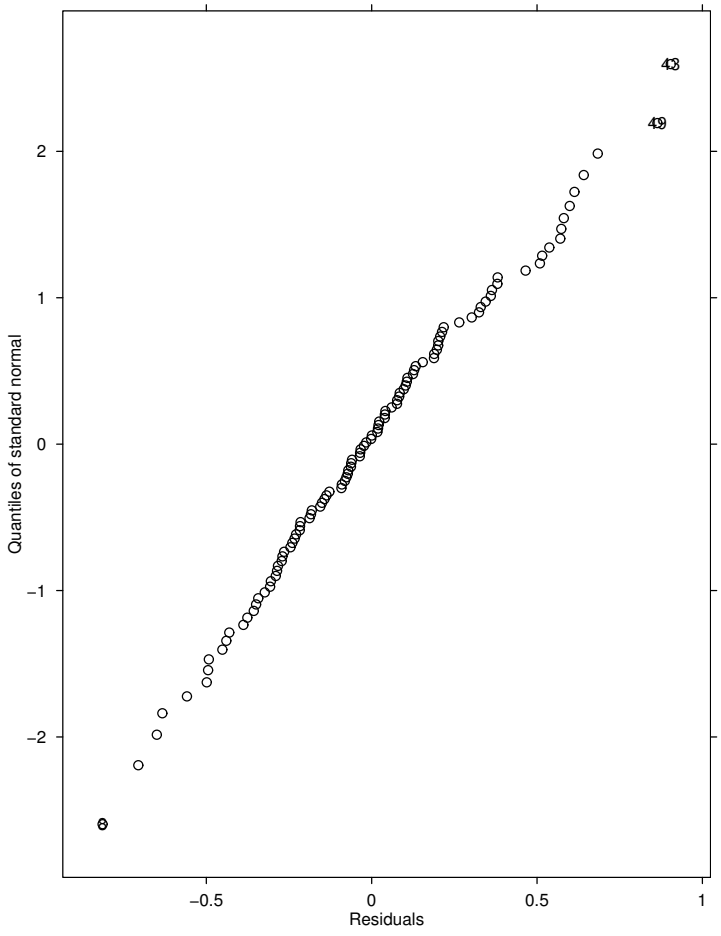


Figure J.6: Residuals against standard normal

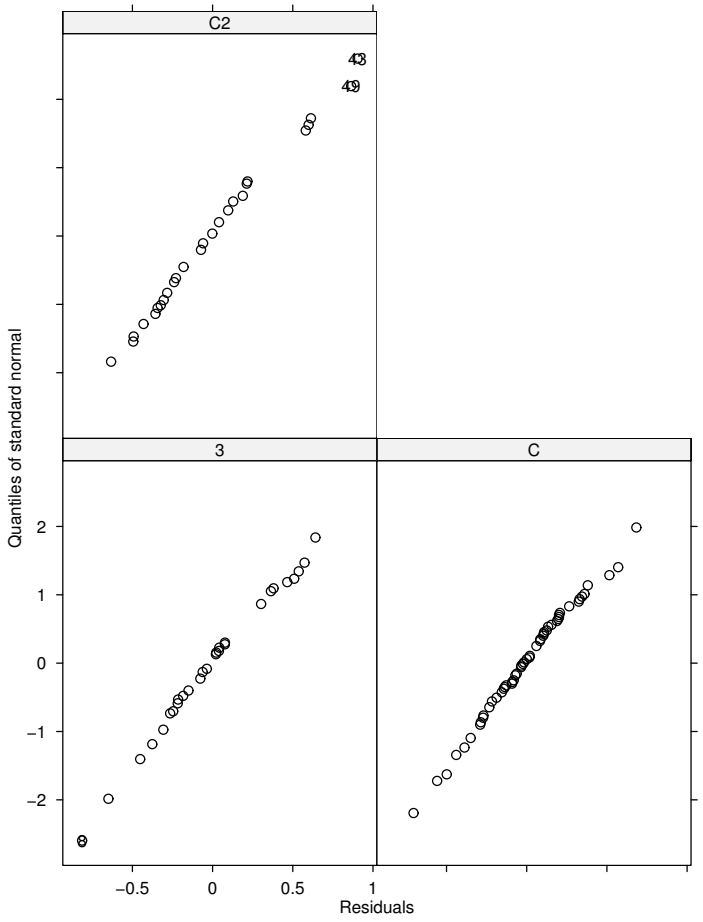


Figure J.7: Residuals against standard normal by group

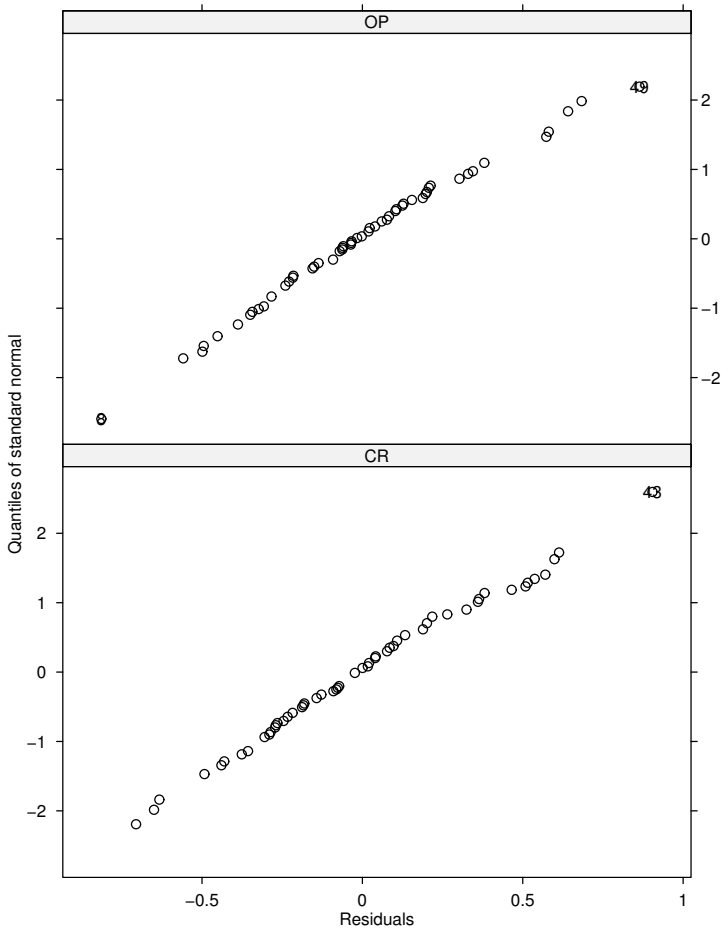


Figure J.8: Residuals against standard normal by domain

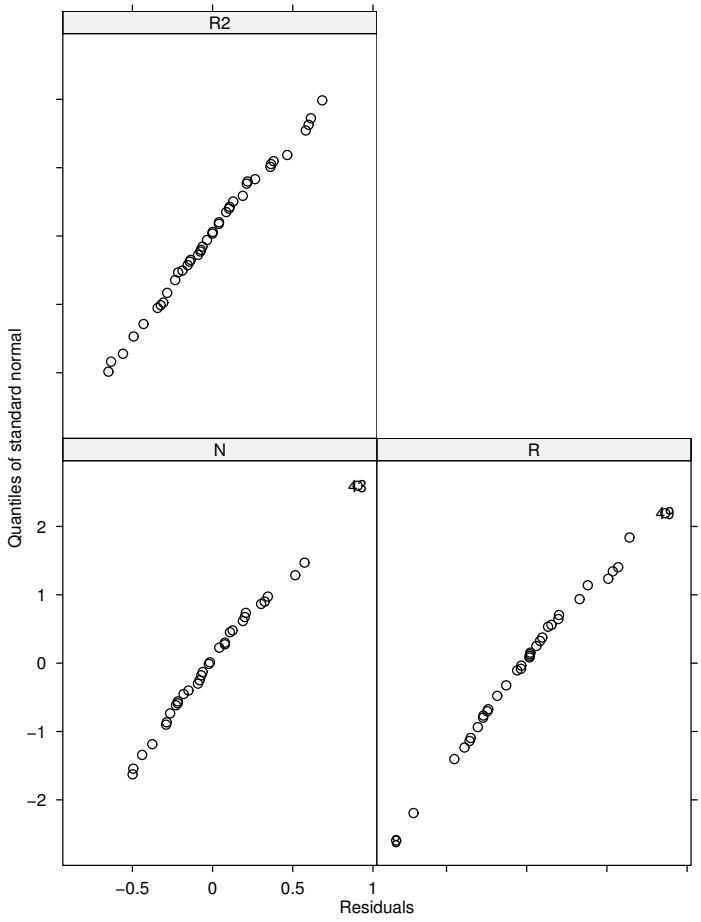


Figure J.9: Residuals against standard normal by type of diagram

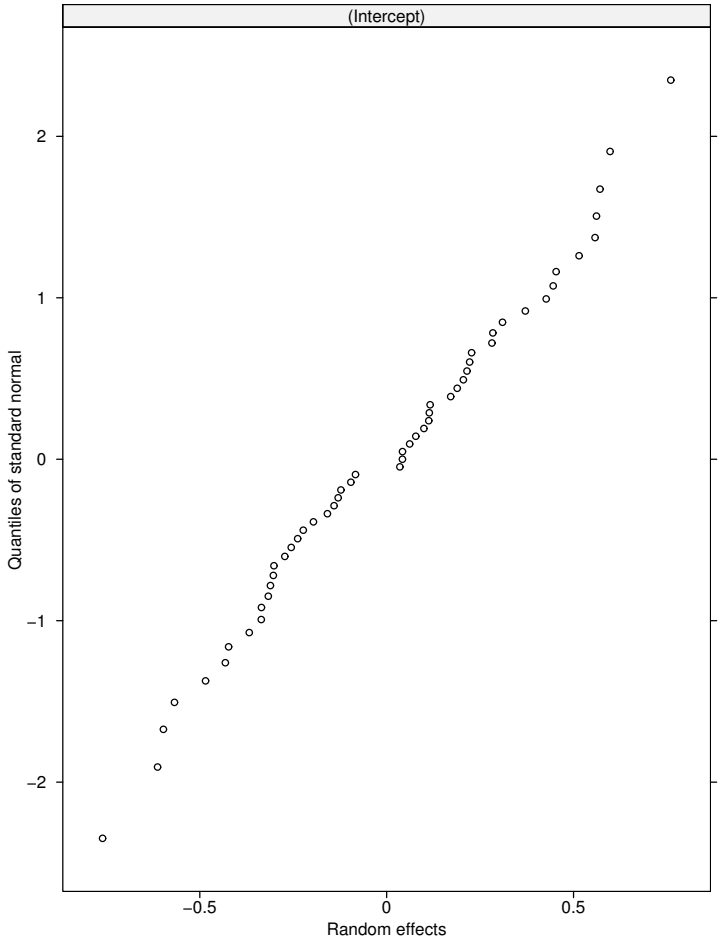


Figure J.10: Random effects against standard normal

```
              Df    Sum Sq   Mean Sq   F value   Pr(>F)
Rules          2     0.264     0.132    0.1242 0.883324
Domain         1 1.053e-31 1.053e-31 9.923e-32 1.000000
Group          2    10.629     5.315    5.0064 0.008655 **
Rules:Domain   2     0.787     0.394    0.3708 0.691208
Rules:Group    4     6.428     1.607    1.5139 0.204723
Domain:Group   2     0.741     0.370    0.3489 0.706374
Residuals     91    96.604     1.062
```

- Usefulness

```
Error: Subject
      Df  Sum Sq Mean Sq
Rules  1 0.72032 0.72032

Error: Within
              Df    Sum Sq   Mean Sq   F value Pr(>F)
Rules          2     2.460     1.230    1.1406 0.3241
Domain         1 9.434e-11 9.434e-11 8.750e-11 1.0000
Group          2     0.414     0.207    0.1921 0.8256
Rules:Domain   2     4.682     2.341    2.1710 0.1199
Rules:Group    4     5.628     1.407    1.3049 0.2742
Domain:Group   2     1.504     0.752    0.6976 0.5004
Residuals     91    98.116     1.078
```

- Ease of Interpretation

```
Error: Subject
      Df  Sum Sq Mean Sq
Rules  1 0.28487 0.28487

Error: Within
              Df    Sum Sq   Mean Sq   F value   Pr(>F)
Rules          2    10.801     5.400    5.1047 0.007922 **
Domain         1 9.434e-11 9.434e-11 8.917e-11 0.999992
Group          2     2.624     1.312    1.2402 0.294167
Rules:Domain   2     1.675     0.837    0.7915 0.456275
Rules:Group    4     5.840     1.460    1.3801 0.247065
Domain:Group   2     0.500     0.250    0.2362 0.790094
Residuals     91    96.272     1.058
```

Appendix K

Scripts for Statistical Analysis

This appendix contains the S-programs (scripts in the S language for use in the R statistics package) used for the statistical analyses in this thesis.

K.1 Card sorting, Round 2, Initial

```
# ##############################################
# Joerg Evermann, 2003
#
# S program for cardsorting analysis
# ##############################################
#
# ##########################
# Define the function for Cohen's Kappa
#
kappaFor2 <- function(r1, r2, na.method = "na.rm")
{
f1 <- ordered(r1)
f2 <- ordered(r2)
levels(f1) <- levels(ordered(c(r1, r2)))
levels(f2) <- levels(ordered(c(r1, r2)))
if (na.method == "na.rm")
na.rm <- T
```

```
else na.rm <- F
ttab <- xtabs( ~ f1+f2, cbind(r1, r2),
drop.unused.levels=FALSE)
tsum <- sum(ttab, na.rm = na.rm)
ttab <- ttab/tsum
tm1 <- apply(ttab, 1, sum, na.rm = na.rm)
tm2 <- apply(ttab, 2, sum, na.rm = na.rm)
agreeP <- sum(diag(ttab), na.rm = na.rm)
chanceP <- sum(tm1 * tm2, na.rm = na.rm)
kappa2 <- (agreeP - chanceP)/(1 - chanceP)
kappaSE <- 1/((1 - chanceP) * sqrt(tsum))
* sqrt(chanceP + chanceP^2
- sum(tm1 * tm2 * (tm1 + tm2), na.rm = na.rm))
kz <- kappa2/kappaSE
kp <- 2 * (1 - pnorm(kz))
ans <- c(kappa2, kappaSE, kz, kp)
names(ans) <- c("kappa", "S.E.", "z.stat", "p.value")
return(ans)
}
#
# ############################
# Read the data file
#
data0 <- read.csv("cardsorting_round2.csv",
header=TRUE, row.names=1)
data <- as.matrix(data0[,c(1:23)])
#
kappas <- c(1:105)*0
count <- 1
#
# ############################
# Compute all pairwise agreements
#
for(i in 1:14){
   for(j in (i+1):15){
      kappas[count] <- kappaFor2(data[i,], data[j,])
      print(kappas[count])
      print(c(count, i, j, kappas[count]))
      count <- count + 1
   }
```

```
}
#
# ############################
# Compute max, mean and min of
# the pairwise agreements
#
kappamax <- max(kappas)
kappaavg <- mean(kappas)
kappamin <- min(kappas)
#
print(c(kappamin, kappamax, kappaavg))
```

K.2 Card sorting, Round 2, After refinement

```
# ################################################
# Joerg Evermann, 2003
#
# S Program for analysis of second round
# cardsorting data
# ################################################
#
# Define function for Cohen's Kappa
#
kappaFor2 <- function(r1, r2, na.method = "na.rm")
{
f1 <- ordered(r1)
f2 <- ordered(r2)
levels(f1) <- levels(ordered(c(r1, r2)))
levels(f2) <- levels(ordered(c(r1, r2)))
if (na.method == "na.rm")
na.rm <- T
else na.rm <- F
ttab <- xtabs( ~ f1+f2, cbind(r1, r2),
drop.unused.levels=FALSE)
tsum <- sum(ttab, na.rm = na.rm)
ttab <- ttab/tsum
tm1 <- apply(ttab, 1, sum, na.rm = na.rm)
tm2 <- apply(ttab, 2, sum, na.rm = na.rm)
```

```
agreeP <- sum(diag(ttab), na.rm = na.rm)
chanceP <- sum(tm1 * tm2, na.rm = na.rm)
kappa2 <- (agreeP - chanceP)/(1 - chanceP)
kappaSE <- 1/((1 - chanceP) * sqrt(tsum))
* sqrt(chanceP + chanceP^2
- sum(tm1 * tm2 * (tm1 + tm2), na.rm = na.rm))
kz <- kappa2/kappaSE
kp <- 2 * (1 - pnorm(kz))
ans <- c(kappa2, kappaSE, kz, kp)
names(ans) <- c("kappa", "S.E.", "z.stat"
, "p.value")
return(ans)
}
#
# ########################
# Read data file
#
data0 <- read.csv("cardsorting_round2.csv",
header=TRUE, row.names=1)
i <- c(1, 2, 4, 5, 7, 9, 10, 11, 13, 14, 15, 16,
17, 19, 20, 21, 23)
data <- as.matrix(data0[,i])
#
# ########################
# Compute all pairwise agreements
#
kappas <- c(1:105)*0
count <- 1

for(i in 1:14){
  for(j in (i+1):15){
    print(kappas[count])
    kappas[count] <- kappaFor2(data[i,], data[j,])
    print(c(count, i, j, kappas[count]))
    count <- count + 1
  }
}
# ########################
# Compute max, mean and min
#
```

```
kappamax <- max(kappas)
kappaavg <- mean(kappas)
kappamin <- min(kappas)
#
print(c(kappamin, kappamax, kappaavg))
```

K.3 Pilot Test

```
# ################################################
# Joerg Evermann, 2003
#
# Compute scale reliabilities from pilot test data
# ################################################
#
# Read data from file
#
data0 <- read.csv("results_fall02.csv", header=TRUE,
row.names=1)
#
# ######################
# Select different proposed
# scales, for two diagrams
# separately
#
i1 <- c("Q1.1", "Q1.2", "Q1.4", "Q1.9", "Q1.10",
"Q1.11", "Q1.13", "Q1.14", "Q1.16")
j1 <- c("Q1.15", "Q1.17", "Q1.19", "Q1.21", "Q1.23")
k1 <- c("Q1.5", "Q1.7", "Q1.20")
#
i2 <- c("Q2.1", "Q2.2", "Q2.4", "Q2.9", "Q2.10",
"Q2.11", "Q2.13", "Q2.14", "Q2.16")
j2 <- c("Q2.15", "Q2.17", "Q2.19", "Q2.21", "Q2.23")
k2 <- c("Q2.5", "Q2.7", "Q2.20")
#
# ######################
# Compute reliabilities
# for each of the six
# scales
```

```
#
data <- as.matrix(data0[,i1])
nv1 <- ncol(data)
alpha <- (nv1/(nv1-1))*(1 - sum(apply(data,2,var))
/var(apply(data,1,sum)))
print(c("Diagram 1, Ease of Interpretation:", alpha))
print(cor(data))

data <- as.matrix(data0[,j1])
nv1 <- ncol(data)
alpha <- (nv1/(nv1-1))*(1 - sum(apply(data,2,var))
/var(apply(data,1,sum)))
print(c("Diagram 1, Usefulness:", alpha))
print(cor(data))

data <- as.matrix(data0[,k1])
nv1 <- ncol(data)
alpha <- (nv1/(nv1-1))*(1 - sum(apply(data,2,var))
/var(apply(data,1,sum)))
print(c("Diagram 1, Information Content:", alpha))
print(cor(data))

data <- as.matrix(data0[,i2])
nv1 <- ncol(data)
alpha <- (nv1/(nv1-1))*(1 - sum(apply(data,2,var))
/var(apply(data,1,sum)))
print(c("Diagram 2, Ease of Interpretation:", alpha))
print(cor(data))

data <- as.matrix(data0[,j2])
nv1 <- ncol(data)
alpha <- (nv1/(nv1-1))*(1 - sum(apply(data,2,var))
/var(apply(data,1,sum)))
print(c("Diagram 2, Usefulness:", alpha))
print(cor(data))

data <- as.matrix(data0[,k2])
nv1 <- ncol(data)
alpha <- (nv1/(nv1-1))*(1 - sum(apply(data,2,var))
/var(apply(data,1,sum)))
```

```
print(c("Diagram 2, Information Content:", alpha))
print(cor(data))

# ######################
# Combine the data from the two diagrams for
# each scale, then  compute reliabilites
# from the combined data set
#
data1 <- as.matrix(data0[, i1])
data2 <- as.matrix(data0[, i2])
nc <- ncol(data1)
nr <- nrow(data1)+nrow(data2)
data3 <- t(array(c(t(data1), t(data2)), dim=c(nc, nr)))
alpha <- (nc/(nc-1))*(1 - sum(apply(data3,2,var))
/var(apply(data3,1,sum)))
print(c("Total, Ease of Interpretation:", alpha))
print(cor(data3))

data1 <- as.matrix(data0[, j1])
data2 <- as.matrix(data0[, j2])
nc <- ncol(data1)
nr <- nrow(data1)+nrow(data2)
data3 <- t(array(c(t(data1), t(data2)), dim=c(nc, nr)))
alpha <- (nc/(nc-1))*(1 - sum(apply(data3,2,var))
/var(apply(data3,1,sum)))
print(c("Total, Usefulness:", alpha))
print(cor(data3))

data1 <- as.matrix(data0[, k1])
data2 <- as.matrix(data0[, k2])
nc <- ncol(data1)
nr <- nrow(data1)+nrow(data2)
data3 <- t(array(c(t(data1), t(data2)), dim=c(nc, nr)))
alpha <- (nc/(nc-1))*(1 - sum(apply(data3,2,var))
/var(apply(data3,1,sum)))
print(c("Total, Information Content:", alpha))
print(cor(data3))

library(mva)
#
```

```
# ######################
# Arrange data for factor analysis
#
i1 <- c("Q1.1", "Q1.2", "Q1.4", "Q1.9", "Q1.10",
"Q1.11", "Q1.13", "Q1.14", "Q1.16", "Q1.15", "Q1.17",
"Q1.19", "Q1.21", "Q1.23", "Q1.5", "Q1.7", "Q1.20")
i2 <- c("Q2.1", "Q2.2", "Q2.4", "Q2.9", "Q2.10",
"Q2.11", "Q2.13", "Q2.14", "Q2.16", "Q2.15", "Q2.17",
"Q2.19", "Q2.21", "Q2.23", "Q2.5", "Q2.7", "Q2.20")

data1 <- data0[, i1]
data2 <- data0[, i2]
nc <- ncol(data1)
nr <- nrow(data1)+nrow(data2)
data3 <- t(array(c(t(data1), t(data2)), dim=c(nc, nr)))

colnames(data3) <- i1

# ########################
# Perform factor extraction with varimax
# rotation for the data for first and
# second diagram and for the combined
# data set
#
fa <- factanal(data3, factors=1, rotation="varimax")
print.factanal(fa, digits=3, cutoff=.1, sort=TRUE)
fa <- factanal(data3, factors=2, rotation="varimax")
print.factanal(fa, digits=3, cutoff=.1, sort=TRUE)
fa <- factanal(data3, factors=3, rotation="varimax")
print.factanal(fa, digits=3, cutoff=.1, sort=TRUE)
```

K.4 Interrater Reliabilities

```
# ################################################
# Joerg Evermann, 2003
#
# Anlysis of interrater agreement for problem
# solving questions
```

```
# ################################################
#
# Define function for Cohen's Kappa
#
kappaFor2 <- function(r1, r2, na.method = "na.rm")
{
f1 <- ordered(r1)
f2 <- ordered(r2)
levels(f1) <- levels(ordered(c(r1, r2)))
levels(f2) <- levels(ordered(c(r1, r2)))
if (na.method == "na.rm")
na.rm <- T
else na.rm <- F
ttab <- xtabs( ~ f1+f2, cbind(r1, r2),
drop.unused.levels=FALSE)
tsum <- sum(ttab, na.rm = na.rm)
ttab <- ttab/tsum
tm1 <- apply(ttab, 1, sum, na.rm = na.rm)
tm2 <- apply(ttab, 2, sum, na.rm = na.rm)
agreeP <- sum(diag(ttab), na.rm = na.rm)
chanceP <- sum(tm1 * tm2, na.rm = na.rm)
kappa2 <- (agreeP - chanceP)/(1 - chanceP)
kappaSE <- 1/((1 - chanceP) * sqrt(tsum))
* sqrt(chanceP + chanceP^2
- sum(tm1 * tm2 * (tm1 + tm2), na.rm = na.rm))
kz <- kappa2/kappaSE
kp <- 2 * (1 - pnorm(kz))
ans <- c(kappa2, kappaSE, kz, kp)
names(ans) <- c("kappa", "S.E.", "z.stat",
"p.value")
return(ans)
}
#
# ########################
# Read the data file
#
data0 <- read.csv("results_for_interrater.csv", header=TRUE)
data <- as.matrix(data0)
#
# ########################
```

```
# Compute kappa for each question
#
kappa1 <- kappaFor2(data[,"CRProb.1.1"], data[,"CRProb.1.2"])
kappa2 <- kappaFor2(data[,"CRProb.2.1"], data[,"CRProb.2.2"])
kappa3 <- kappaFor2(data[,"CRProb.3.1"], data[,"CRProb.3.2"])
kappa4 <- kappaFor2(data[,"CRProb.4.1"], data[,"CRProb.4.2"])
kappa5 <- kappaFor2(data[,"CRProb.5.1"], data[,"CRProb.5.2"])
kappa6 <- kappaFor2(data[,"OPProb.1.1"], data[,"OPProb.1.2"])
kappa7 <- kappaFor2(data[,"OPProb.2.1"], data[,"OPProb.2.2"])
kappa8 <- kappaFor2(data[,"OPProb.3.1"], data[,"OPProb.3.2"])
kappa9 <- kappaFor2(data[,"OPProb.4.1"], data[,"OPProb.4.2"])
kappa10 <- kappaFor2(data[,"OPProb.5.1"], data[,"OPProb.5.2"])
kappa11 <- kappaFor2(data[,"OPProb.6.1"], data[,"OPProb.6.2"])
kappa12 <- kappaFor2(data[,"OPProb.7.1"], data[,"OPProb.7.2"])
#
# ############################
# Print the kappa values
#
print(kappa1)
print(kappa2)
print(kappa3)
print(kappa4)
print(kappa5)
print(kappa6)
print(kappa7)
print(kappa8)
print(kappa9)
print(kappa10)
print(kappa11)
print(kappa12)
```

K.5 Scale Reliabilities

labelapp:scales.R

```
# ################################################
# Joerg Evermann, 2003
#
```

```
# Scale reliabilities from the final data set
# ################################################
#
# Read data from file and separate the scales
#
data0 <- read.csv("results_for_scales.csv",
header=TRUE)
i1 <- c("CRC.1", "CRC.2", "CRC.4", "CRC.9", "CRC.10",
"CRC.11", "CRC.13", "CRC.14", "CRC.16")
j1 <- c("CRC.15", "CRC.17", "CRC.19", "CRC.21", "CRC.23")
k1 <- c("CRC.5", "CRC.7", "CRC.20")

i2 <- c("OPC.1", "OPC.2", "OPC.4", "OPC.9", "OPC.10",
"OPC.11", "OPC.13", "OPC.14", "OPC.16")
j2 <- c("OPC.15", "OPC.17", "OPC.19", "OPC.21", "OPC.23")
k2 <- c("OPC.5", "OPC.7", "OPC.20")
#
# ########################
# Compute Cronbach's alpha for the six scales
#
data <- as.matrix(data0[,i1])
nv1 <- ncol(data)
alpha <- (nv1/(nv1-1))*(1 - sum(apply(data,2,var))
/var(apply(data,1,sum)))
print(c("Car Rental, Ease of Interpretation:", alpha))
#
data <- as.matrix(data0[,j1])
nv1 <- ncol(data)
alpha <- (nv1/(nv1-1))*(1 - sum(apply(data,2,var))
/var(apply(data,1,sum)))
print(c("Car Rental, Usefulness:", alpha))
#
data <- as.matrix(data0[,k1])
nv1 <- ncol(data)
alpha <- (nv1/(nv1-1))*(1 - sum(apply(data,2,var))
/var(apply(data,1,sum)))
print(c("Car Rental, Information Content:", alpha))
#
data <- as.matrix(data0[,i2])
nv1 <- ncol(data)
```

```
alpha <- (nv1/(nv1-1))*(1 - sum(apply(data,2,var))
/var(apply(data,1,sum)))
print(c("Order Processing, Ease of Interpretation:",
alpha))
#
data <- as.matrix(data0[,j2])
nv1 <- ncol(data)
alpha <- (nv1/(nv1-1))*(1 - sum(apply(data,2,var))
/var(apply(data,1,sum)))
print(c("Order Processing, Usefulness:", alpha))
#
data <- as.matrix(data0[,k2])
nv1 <- ncol(data)
alpha <- (nv1/(nv1-1))*(1 - sum(apply(data,2,var))
/var(apply(data,1,sum)))
print(c("Order Processing, Information Content:",
alpha))
#
# ########################
# Combine data across domains
# Compute reliabilities across complete data set
#
data1 <- as.matrix(data0[, i1])
data2 <- as.matrix(data0[, i2])
nc <- ncol(data1)
nr <- nrow(data1)+nrow(data2)
data3 <- t(array(c(t(data1), t(data2)), dim=c(nc, nr)))
alpha <- (nc/(nc-1))*(1 - sum(apply(data3,2,var))
/var(apply(data3,1,sum)))
print(c("Total, Ease of Interpretation:", alpha))
#
data1 <- as.matrix(data0[, j1])
data2 <- as.matrix(data0[, j2])
nc <- ncol(data1)
nr <- nrow(data1)+nrow(data2)
data3 <- t(array(c(t(data1), t(data2)), dim=c(nc, nr)))
alpha <- (nc/(nc-1))*(1 - sum(apply(data3,2,var))
/var(apply(data3,1,sum)))
print(c("Total, Usefulness:", alpha))
#
```

```
data1 <- as.matrix(data0[, k1])
data2 <- as.matrix(data0[, k2])
nc <- ncol(data1)
nr <- nrow(data1)+nrow(data2)
data3 <- t(array(c(t(data1), t(data2)), dim=c(nc, nr)))
alpha <- (nc/(nc-1))*(1 - sum(apply(data3,2,var))
/var(apply(data3,1,sum)))
print(c("Total, Information Content:", alpha))
#
library(mva)
#
# ############################
# Combine the item sets from the three scales
# for factor extraction
#
i1 <- c("CRC.1", "CRC.2", "CRC.4", "CRC.9", "CRC.10",
"CRC.11", "CRC.13", "CRC.14", "CRC.16", "CRC.15", "CRC.17",
"CRC.19", "CRC.21", "CRC.23", "CRC.5", "CRC.7", "CRC.20")
i2 <- c("OPC.1", "OPC.2", "OPC.4", "OPC.9", "OPC.10",
"OPC.11", "OPC.13", "OPC.14", "OPC.16", "OPC.15", "OPC.17",
"OPC.19", "OPC.21", "OPC.23", "OPC.5", "OPC.7", "OPC.20")
#
data1 <- data0[, i1]
data2 <- data0[, i2]
nc <- ncol(data1)
nr <- nrow(data1)+nrow(data2)
data3 <- t(array(c(t(data1), t(data2)), dim=c(nc, nr)))
#
colnames(data3) <- i1
#
# ############################
# Extract the Bartlett factor scores for each domain
#
#
print("Factor Analysis Car Rental")
#
fa <- factanal(data1, factors=3, rotation="varimax",
scores="Bartlett")
print.factanal(fa, digits=3, cutoff=.1, sort=TRUE)
print(fa$scores)
```

```
#
print ("Factor Analysis Order Processing")
#
fa <- factanal(data2, factors=3, rotation="varimax",
scores="Bartlett")
print.factanal(fa, digits=3, cutoff=.1, sort=TRUE)
print(fa$scores)
#
print ("Factor Analysis Combined")
#
fa <- factanal(data3, factors=3, rotation="varimax",
scores="Bartlett")
print.factanal(fa, digits=3, cutoff=.1, sort=TRUE)
print(fa$scores)
```

K.6 Descriptive Statistics

```
# ################################################
# Joerg Evermann, 2003
#
# S program for factor level means
#
# ################################################
#
# ######################
# Read the data from file
#
data0 <- read.csv(file="results_for_anova2.csv",
header=TRUE)
attach(data0)
#
# ################################################
# ANOVA First
#
# ####################
# Some data plots
#
```

```
tapply(Prob, factor(data0$Rules), mean)
tapply(Prob, factor(data0$Domain), mean)
tapply(Prob, factor(data0$Group), mean)
tapply(Prob, factor(data0$Rules), sd)
tapply(Prob, factor(data0$Domain), sd)
tapply(Prob, factor(data0$Group), sd)

tapply(Comp, factor(data0$Rules), mean)
tapply(Comp, factor(data0$Domain), mean)
tapply(Comp, factor(data0$Group), mean)
tapply(Comp, factor(data0$Rules), sd)
tapply(Comp, factor(data0$Domain), sd)
tapply(Comp, factor(data0$Group), sd)

tapply(Interpretation, factor(data0$Rules), mean)
tapply(Interpretation, factor(data0$Domain), mean)
tapply(Interpretation, factor(data0$Group), mean)
tapply(Interpretation, factor(data0$Rules), sd)
tapply(Interpretation, factor(data0$Domain), sd)
tapply(Interpretation, factor(data0$Group), sd)

tapply(Usefulness, factor(data0$Rules), mean)
tapply(Usefulness, factor(data0$Domain), mean)
tapply(Usefulness, factor(data0$Group), mean)
tapply(Usefulness, factor(data0$Rules), sd)
tapply(Usefulness, factor(data0$Domain), sd)
tapply(Usefulness, factor(data0$Group), sd)

tapply(Information, factor(data0$Rules), mean)
tapply(Information, factor(data0$Domain), mean)
tapply(Information, factor(data0$Group), mean)
tapply(Information, factor(data0$Rules), sd)
tapply(Information, factor(data0$Domain), sd)
tapply(Information, factor(data0$Group), sd)
```

K.7 Hypothesis Testing

```
# ###############################################
```

```
# Joerg Evermann, 2003
#
# S program for hypothesis testing
#
# ################################################
#
# ######################
# Load NLME library
#
library(nlme)
#
# ######################
# Read the data from file
#
data0 <- read.csv(file="results_for_anova2.csv",
header=TRUE)
attach(data0)
#
# ################################################
# ANOVA First
#
# ####################
# Some data plots
#
postscript("aov1b.eps", width=6, onefile=FALSE)
par ( mfrow=c(3,1) )
plot.factor(data0$Rules, Prob)
plot.factor(data0$Domain, Prob)
plot.factor(data0$Group, Prob)
par ( mfrow=c(1,1) )
dev.off()
#
# ####################
# Now the model fit
#
fitted.ao1 <- aov(Prob ~ (Rules + Domain + Group)^2 +
UML.TTL + UML.A + SelfAssess + Time + Comp +
Interpretation + Usefulness + Information +
Error(Subject), data0)
#
```

```
summary(fitted.ao1)
coef(fitted.ao1)
resid(fitted.ao1)
#
# ####################
# Interaction plots
#
postscript("aov1c.eps", width=6, onefile=FALSE)
interaction.plot(Rules, Group, Prob)
dev.off()
postscript("aov1d.eps", width=6, onefile=FALSE)
interaction.plot(Rules, Domain, Prob)
dev.off()
#
#
#
# ###############
# Smaller model
#
fitted.ao2 <- aov(Prob ~ (Rules + Group)^2 +
UML.TTL + Error(Subject), data0)
#
summary(fitted.ao2)
coef(fitted.ao2)
resid(fitted.ao2)
#
#
# ################################################
# Linear Models
#
# Build the groupedData object
#
gdform = Prob ~ Rules | Subject
outer = ~ UML.A * UML.TTL * Rules * Group
inner = ~ Interpration * Usefulness * Information
* Comp * Time * Domain
gData = groupedData(formula = gdform, data = data0,
outer = outer, inner = inner)
#
# ################################################
```

```
# Analysis using linear models
#
# ########################
# Plot the data
#
postscript("data1.eps", width=6, onefile=FALSE)
plot(gData)
dev.off()
postscript("data2.eps", width=6, onefile=FALSE)
plot(gData, outer = ~ Domain * Group)
dev.off()
#
# ################################################
# Try a regular linear model first
#
fitted.lm0 <- lm(Prob ~ (Rules + Domain + Group)^2 +
UML.TTL + UML.A + Time + Information +
Interpretation + Usefulness + Comp + SelfAssess, gData)
summary(fitted.lm0)
#
# ########################
# Diagnostic plots
#
par( mfrow=c(3,2) )
postscript("diaglm0.eps", width=6, onefile=FALSE)
plot (fitted.lm0)
dev.off()
#
#
# ####################
# Use the auto step function to add one factor at
# a time until the model does not improve anymore
#
fitted.lm.base <- lm(Prob ~ 1, gData)
#
step(fitted.lm.base, Prob ~ (Rules + Domain + Group)^2
+ UML.TTL + UML.A + Time + Information + Interpretation
+ Usefulness + Comp + SelfAssess)
#
# ######################
```

```
# Try a smaller model
#
fitted.lm1 <- lm(Prob ~ (Rules + Group)^2 + UML.TTL, gData)
summary(fitted.lm1)
#
par( mfrow=c(3,2) )
postscript("diaglm1.eps", width=6, onefile=FALSE)
plot (fitted.lm1)
dev.off()
#
#
# ################################################
# Fit the model using linear mixed effects modelling
#
fitted1 <- lme(Prob ~ (Rules + Domain + Group)^2
+ UML.TTL + UML.A + Time + Information
+ Interpretation + Usefulness + Comp + SelfAssess,
gData, random = ~ 1 | Subject)
summary(fitted1)
anova(fitted1)
intervals(fitted1)
#
#
# ######################
# Plot diagnositics
#
postscript("observed1.eps", width=6, onefile=FALSE)
plot(fitted1, Prob ~ fitted(.))
dev.off()
postscript("resid1a.eps", width=6, onefile=FALSE)
plot(fitted1, Subject~resid(.), abline= 0)
dev.off()
postscript("resid1b.eps", width=6, onefile=FALSE)
plot(fitted1, resid(.) ~ fitted(.) | Group)
dev.off()
postscript("resid1c.eps", width=6, onefile=FALSE)
plot(fitted1, resid(.) ~ fitted(.) | Domain)
dev.off()
postscript("resdi1d.eps", width=6, onefile=FALSE)
plot(fitted1, resid(.) ~ fitted(.) | Rules)
```

```
dev.off()
#
#
# ########################
# Test for heteroscedasticity:
#
# Model different error variances between the
# different factor stratums, e.g. Group,
# Domain and Rules
#
# Test with ANOVA Model
#
fitted1a <- update (fitted1,
weights=varIdent(form = ~ 1 | Group))
anova(fitted1, fitted1a)
#
fitted1b <- update (fitted1,
weights=varIdent(form = ~ 1 | Domain))
anova(fitted1, fitted1b)
#
fitted1c <- update (fitted1,
weights=varIdent(form = ~ 1 | Rules))
anova(fitted1, fitted1c)
#
# ##########################
# Plot diagnostics for standard errors/residuals
#
postscript("qqnorm1a.eps", width=6, onefile=FALSE)
qqnorm(fitted1, ~resid(.), id = 0.1)
dev.off()
postscript("qqnorm1b.eps", width=6, onefile=FALSE)
qqnorm(fitted1, ~ resid(.) | Group, id = 0.1)
dev.off()
postscript("qqnorm1c.eps", width=6, onefile=FALSE)
qqnorm(fitted1, ~ resid(.) | Domain, id = 0.1)
dev.off()
postscript("qqnorm1d.eps", width=6, onefile=FALSE)
qqnorm(fitted1, ~ resid(.) | Rules, id = 0.1)
dev.off()
#
```

```
############################
# Test for independence of random effects
#
# Random effects are normally distributed and
# have zero mean
#
postscript("ranef1.eps", width = 6, onefile=FALSE)
qqnorm(fitted1, ~ranef(.), id = 0.10, cex = 0.7)
dev.off()
#
#
# ############################
# Try to find the best, most parsimonious model
# by adding in terms
#
# Start with a simple model
#
fitted2 <- lme(Prob ~ (Rules + Group)^2, gData,
random = ~ 1 | Subject)
summary(fitted2)
anova(fitted2)
intervals(fitted2)
#
############################
# Plot diagnostics
#
postscript("observed2.eps", width=6, onefile=FALSE)
plot(fitted2, Prob ~ fitted(.))
dev.off()
#
# ############################
# Test again for heteroscedasticity:
#
# Model different error variances between the different
# factor stratums, e.g. Group, Domain and Rules
#
# Test with ANOVA Model
#
fitted2a <- update (fitted2,
weights=varIdent(form = ~ 1 | Group))
```

```
anova(fitted2, fitted2a)
#
fitted2b <- update (fitted2,
weights=varIdent(form = ~ 1 | Rules))
anova(fitted2, fitted2b)
#
#
postscript("resid2a.eps", width=6, onefile=FALSE)
plot(fitted1, Subject~resid(.), abline= 0)
dev.off()
postscript("resid2b.eps", width=6, onefile=FALSE)
plot(fitted1, resid(.) ~ fitted(.) | Group)
dev.off()
postscript("resid2c.eps", width=6, onefile=FALSE)
plot(fitted1, resid(.) ~ fitted(.) | Rules)
dev.off()
#
postscript("qqnorm2a.eps", width=6, onefile=FALSE)
qqnorm(fitted2, ~resid(.))
dev.off()
postscript("qqnorm2b.eps", width=6, onefile=FALSE)
qqnorm(fitted2, ~ resid(.) | Group)
dev.off()
postscript("qqnorm2c.eps", width=6, onefile=FALSE)
qqnorm(fitted2, ~ resid(.) | Rules)
dev.off()
#
# ############################
# Test again for independence of random effects
#
# Random effects are normally distributed and
# have zero mean
#
postscript("ranef2.eps", width = 6, onefile=FALSE)
qqnorm(fitted2, ~ranef(.), id = 0.10, cex = 0.7)
dev.off()
#
# ############################
# Add some more variables to see...
#
```

```
fitted3 <- lme(Prob ~ (Rules + Domain + Group)^2,
gData, random = ~ 1 | Subject)
summary(fitted3)
anova(fitted3)
#
fitted4 <- lme(Prob ~ (Rules + Domain + Group)^2
+ UML.TTL, gData,random =  ~ 1 | Subject)
summary(fitted4)
anova(fitted4)
#
fitted5 <- lme(Prob ~ (Rules + Domain + Group)^2
+ UML.TTL + Time, gData, random = ~ 1 | Subject)
summary(fitted5)
anova(fitted5)
#
fitted6 <- lme(Prob ~ (Rules + Domain + Group)^2
+ UML.TTL + Time + Comp, gData, random = ~ 1 | Subject)
summary(fitted6)
anova(fitted6)
#
fitted7 <- lme(Prob ~ (Rules + Domain + Group)^2
+ UML.TTL + Time + Comp + SelfAssess, gData,
random = ~ 1 | Subject)
summary(fitted7)
anova(fitted7)
#
fitted8 <- lme(Prob ~ (Rules + Domain + Group)^2
+ UML.TTL + Time + Comp + SelfAssess + UML.A
+ Information + Usefulness + Interpretation,
gData, random = ~ 1 | Subject)
summary(fitted8)
anova(fitted8)
#
```

K.8 Diagram Properties

```
# ##############################################
# Joerg Evermann, 2003
```

```
#
# Anova procedure for analyzing the effect of
# Rules, Domain and Group on Information content,
# Usefulness and Ease of interpretation
# ###############################################
#
# ########################
# Read the data from file
#
data0 <- read.csv(file="results_for_anova2.csv",
header=TRUE)
attach(data0)
#
# ########################
# ANOVA models for each
# of the perceived control
# variables
#
fitted.me1 <-
aov(Information ~ (Rules + Domain + Group)^2
+ Error(Subject), data0)
fitted.me2 <-
aov(Usefulness ~ (Rules + Domain + Group)^2
+ Error(Subject), data0)
fitted.me3 <-
aov(Interpretation ~ (Rules + Domain + Group)^2
+ Error(Subject), data0)
#
summary(fitted.me1)

tapply(Information, factor(data0$Rules), mean)
tapply(Information, factor(data0$Domain), mean)
tapply(Information, factor(data0$Group), mean)
tapply(Information, factor(data0$Rules), sd)
tapply(Information, factor(data0$Domain), sd)
tapply(Information, factor(data0$Group), sd)

#
summary(fitted.me2)
```

```
tapply(Usefulness, factor(data0$Rules), mean)
tapply(Usefulness, factor(data0$Domain), mean)
tapply(Usefulness, factor(data0$Group), mean)
tapply(Usefulness, factor(data0$Rules), sd)
tapply(Usefulness, factor(data0$Domain), sd)
tapply(Usefulness, factor(data0$Group), sd)

#
summary(fitted.me3)

tapply(Interpretation, factor(data0$Rules), mean)
tapply(Interpretation, factor(data0$Domain), mean)
tapply(Interpretation, factor(data0$Group), mean)
tapply(Interpretation, factor(data0$Rules), sd)
tapply(Interpretation, factor(data0$Domain), sd)
tapply(Interpretation, factor(data0$Group), sd)

#
# ############################
# Determine effect of group on time and
# UML knowledge
#
fitted.me4 <- aov(Time ~ Group + Error(Subject),
data0)
summary(fitted.me4)

tapply(Time, factor(data0$Rules), mean)
tapply(Time, factor(data0$Domain), mean)
tapply(Time, factor(data0$Group), mean)
tapply(Time, factor(data0$Rules), sd)
tapply(Time, factor(data0$Domain), sd)
tapply(Time, factor(data0$Group), sd)

#
fitted.me5 <- aov(UML.TTL ~ Group + Error(Subject),
data0)
summary(fitted.me5)

tapply(UML.TTL, factor(data0$Rules), mean)
tapply(UML.TTL, factor(data0$Domain), mean)
```

```
tapply(UML.TTL, factor(data0$Group), mean)
tapply(UML.TTL, factor(data0$Rules), sd)
tapply(UML.TTL, factor(data0$Domain), sd)
tapply(UML.TTL, factor(data0$Group), sd)
```