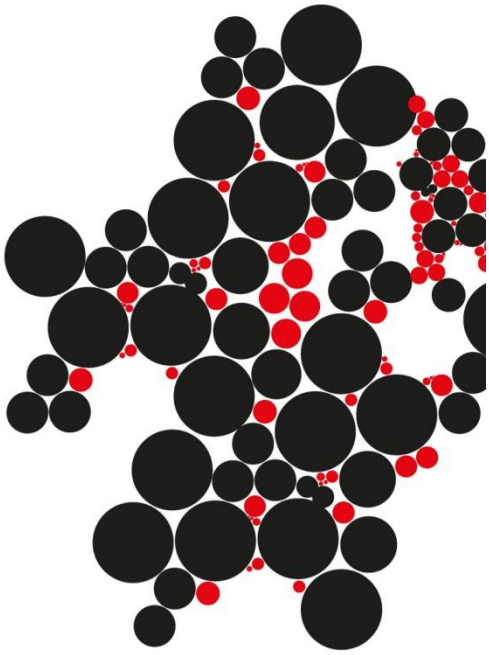
MASTER THESIS



# BEHAVIOR SPECIFICATION FOR ONTOLOGICALLY GROUNDED CONCEPTUAL MODELS


Ruud Wiegers

COMPUTER SCIENCE

SERVICES, CYBERSECURITY AND SAFETY

**SUPERVISORS**

dr. Luís Ferreira Pires

dr. João Paulo Almeida

dr.ir. Rom Langerak



UNIVERSITY OF TWENTE.

02.11.2014

## Abstract

Conceptual modeling of a domain of interest is an important step in the design of information systems. To facilitate such an approach, suitable modeling languages have been developed, among which is OntoUML, a philosophically well-founded conceptual modeling language based on UML. However, OntoUML can be used to describe the structure of a domain, but it has not been designed to support the description of behavior.

This thesis discusses the development a language, named OBSL, for modeling behaviors of elements of a domain, complementary to a structural specification. OBSL is designed to be expressive enough to cover most common behaviors, but the focus is placed on ease of use. It describes behavior in terms of transitions between states, an abstraction that avoids the use of temporal logic. A graphical notation for OBSL is developed, according to a theory for diagram design aimed at cognitive effectiveness, resulting in diagrams that are intuitively appealing for modelers, and therefore relatively easy to understand.

The semantics of OBSL is expressed in Alloy, a logic-based modeling language that has also been used to express the semantics of OntoUML. An OBSL specification combined with its OntoUML model can be translated to an Alloy specification, which can then be used to simulate examples of the specified behaviors.

The language, graphical notation and transformation to Alloy are supported by a tool set, consisting of a graphical editor that can be used to create and modify behavior specifications, as well as to generate a representation in Alloy.

# Acknowledgements

This thesis could not have been completed without the support of many people. I would like to thank the following people in particular:

Luís Ferreira Pires and João Paulo Andrade Almeida, my supervisors, who through many insightful Skype conversations have provided the feedback and guidance I required. Rom Langerak, my secondary supervisor, who has always shown enthusiasm and interest, even though this thesis lies outside his personal area of research. John, Tiago, and anyone else at NEMO who answered my questions about OntoUML and related material. Ivan Kurtev, who supervised my research topics, for sparking my interest in Model Driven Engineering, which started me on the road that led to this thesis. My parents Frans and Mirjam, and my sister Ineke, for their patience and quiet support. And finally, Irana, whose loving support has helped me through every phase of the lengthy process of completing my studies.

Thank you.

Ruud Wiegers

# Table of contents

# 1 Introduction

This chapter introduces the topic of this thesis. Section 1.1 provides the context and motivation for our work. Section 1.2 describes the goals of our work, and Section 1.3 introduces the approach to meet these goals. Section 1.4 outlines the structure of the thesis.

## 1.1 Motivation

The understanding and communication of ideas is essential in every field of science and engineering. The process of capturing an abstraction of the world around us in a model, for purposes of understanding and communication regarding a specific domain, is called conceptual modeling [1].

Conceptual modeling plays an essential role in many areas of computer science, such as database and information systems design, software and domain engineering, design of knowledge-based systems, requirements engineering, information integration, semantic interoperability, natural language processing, and enterprise modeling [2].

Because conceptual models play such a key role, the quality of conceptual models is vital. There are two important measures of the quality of a conceptual model: the *suitability* of the abstractions of the conceptual model to the task of representing relevant aspects of reality, and the *effectiveness* with which the model can *communicate* its abstractions to people [3]. Using a language specifically engineered for the purpose of creating conceptual models gives the modeler the tools he needs to achieve the desired level of quality. In [4], it is argued that by basing a conceptual modeling language on a reference foundational ontology (a domain-independent description of reality), the accuracy of the resulting conceptual models is improved.

OntoUML is an example of such a conceptual modeling language. It consists of a redesign of a subset of UML [5] according to a reference ontology for conceptual modeling called the Unified Foundational Ontology, UFO [4]. It is philosophically and cognitively well-founded, which helps improve the soundness and completeness of conceptual models constructed using this language.

OntoUML initially focused on the structural aspects of conceptual modeling: it describes different types of entities, their properties and parts, and the relations that connect them. OntoUML classification is based (among other things) on the metaproperties found in OntoClean [6]. Some of these metaproperties deal with *possibility* or alethic modality. This means that while OntoUML does not include a way to explicitly express dynamic concepts (e.g. modeling of events and processes), it does possess a limited capacity for describing dynamic aspects of modeled entities (defining ways in which entities may change over time).

The tools supporting OntoUML include a model simulator, which aids validation of conceptual models by generating example *instances*: configurations of entities that are valid according to the conceptual model [7]. This aids a modeler during the development process, and it also facilitates *communication* about the conceptual model by automatically generating examples. These examples are intended to demonstrate the properties of entities of a conceptual model, including the alethic modal properties. To this end, an example is not a single world, but comprises a temporal structure, with

multiple worlds representing past, future, and *possible* worlds. The resulting simulations are sufficient to show alethic properties of domain elements.

The use of complete temporal structures as examples invites a modeler to read it not only as a representation of the alethic properties of the structural conceptual model, but as a representation of the dynamic aspects of the domain represented by the conceptual model. The lack of expressivity of OntoUML regarding these dynamic aspects means it is not possible to describe the *behavior* of the domain entities in the conceptual model in sufficient detail. For example, consider a domain which includes people, which may be either *children* or *adults*. OntoUML is capable of modeling these concepts, but it cannot express the domain-specific constraint that persons cannot be children after being adults. Therefore, example models may show a person as an adult in one moment, and as a child the next. Such an example model *appears* to represent behavior, and specifically behavior that does not match the domain being modeled.

## 1.2 Goals

This thesis aims to remedy the limitations of OntoUML concerning the representation of behaviors by developing a language for modeling the dynamic aspects of elements of a domain. The resulting behavioral models will be complementary to conceptual models created with OntoUML. The formalization of these behavioral models will be compatible with the OntoUML models they are based on, and will be usable by the model simulator when generating examples. In this way, the purpose of the model simulator is extended: an example model will not only represent the structural and alethic properties of entities in a domain, but will also include their dynamic aspects.

As a secondary objective, this thesis aims to examine the benefits of developing a language that prioritizes *cognitive effectiveness* over complete expressivity. While the language should be expressive enough to model most behavior, the focus is on usability by modelers. We explore three facets to achieve this. First, we aim for a language with semantics that can be understood in abstract terms, without necessarily requiring the use of formal logic. Second, we explore the use of a visual notation to achieve a concise and clear representation of our language. Third, we examine how the use of model simulations can aid the modeler in creating behavioral specifications.

In summary, the research goals of this thesis are:

1. To identify how dynamic aspects of a domain can be represented in a conceptual model, in a way that is complementary to a (static) conceptual model.
    a. To describe the distinction between static and dynamic aspects in conceptual modeling.
    b. To describe the limits of OntoUML with respect to describing dynamic aspects.
    c. To define a modeling language capable of modeling these aspects in complementary to static conceptual models defined in OntoUML.
2. To examine methods by which the *cognitive effectiveness* of a conceptual modeling language can be enhanced.
    a. To examine the effectiveness of a language which has limited expressivity (i.e., can we define a language that do not require the user to understand formal logic?)
    b. To examine the advantages of a suitable visual notation.
    c. To describe ways in which integration with OntoUML's infrastructure, particularly the model simulator, can be used to enhance the modeling process.
3. To create a prototype as a proof-of-concept for the language and the elements of its design.
    a. Develop a visual editor.
    b. Develop integration with OntoUML domain models and their simulation with the model simulator.

## 1.3 Approach

In order to achieve the goals as presented, we take the following approach:

1. To identify how dynamic aspects of conceptual models can be represented, we first examine the existing work on conceptual modeling, focusing on OntoUML, and its model simulator in particular.
2. The results of this analysis are used to design a modeling language for the behavior of conceptual models. Integration with OntoUML is achieved by basing the formal semantics of our modeling language on the semantics defined for OntoUML.
3. To create an appropriate visual representation for this language, we utilize a theory for visual notations developed in [8].
4. To validate our approach, we create an implementation of the modeling language in the Eclipse Modeling Framework [9]. This tool is then used to specify and simulate behaviors for a domain ontology in a case study.

## 1.4 Structure

This thesis is further structured as follows: Chapter 1.2 presents the theoretical foundations of conceptual modeling. Chapter 3 presents the structure of UFO, the foundation of OntoUML. Chapter 4 discusses the model simulator and the issues with respect to dynamic aspects of conceptual models in OntoUML. Chapter 5 presents the language we developed to allow the specification of behaviors of OntoUML models. Chapter 6 discusses the graphical notation for this behavior specification language. Chapter 5.5 describes a tool to create, manage and simulate behavior specifications. Chapter 7 evaluates the approach by means of a case study.

Figure 1 summarizes the structure of this thesis.

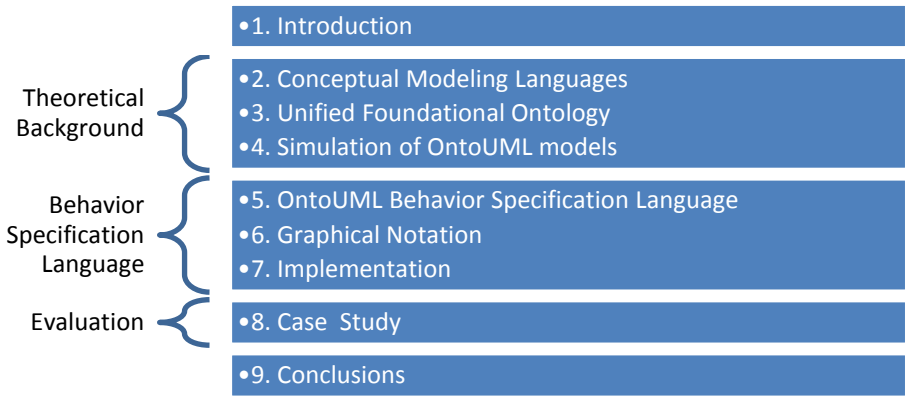



**Figure 1: Thesis structure**

# 2 Conceptual Modeling Languages

Conceptual modeling in a broad sense is the practice of creating a description of aspects of reality for the purpose of understanding and communication. It applies to many fields of engineering, and it touches on the study of language, cognition, and philosophy. This broad array of sources and applications makes for a variety of definitions and terminology. For the purpose of clarity, this chapter establishes the definitions and terminology regarding conceptual modeling that is used in this thesis.

Section 2.1 describes the core concepts of conceptual modeling. Section 2.2 discusses the different levels of abstraction in conceptual models. Section 2.3 discusses the key criteria that determine the quality of a conceptual model. Section 2.5 discusses other criteria that influence conceptual modeling languages. Section 2.5 describes the characteristics of OntoUML with respect to the established framework and criteria.

## 2.1 Defining conceptual modeling

A suitable definition of conceptual modeling is given by Mylopoulos [1], who defines it as: "the activity of formally describing some aspects of the physical and social world around us for purposes of understanding and communication. Moreover, it supports structuring and inferential facilities that are psychologically grounded. After all, the descriptions that arise from conceptual modeling activities are intended to be used by humans, not machines".

This definition highlights several aspects that are crucial to our understanding of conceptual modeling. Conceptual modeling is concerned with making descriptions. Conceptual models are not merely thoughts; they are represented in some tangible form. Further, conceptual models are descriptions of aspects of the world. A conceptual model is a description of an abstraction of reality. Different conceptual models can describe a situation in different ways, according to their purpose. For example, medical doctors use different conceptual models of people than accountants. Conceptual models are models of a specific domain of discourse.

Combining these aspects, conceptual models can be described as "a representation of an abstraction of reality". This definition of conceptual modeling is derived from the so-called 'triangle of meaning' or 'triangle of reference' (Figure 2), which is a foundational concept in the study of language and cognition [10].

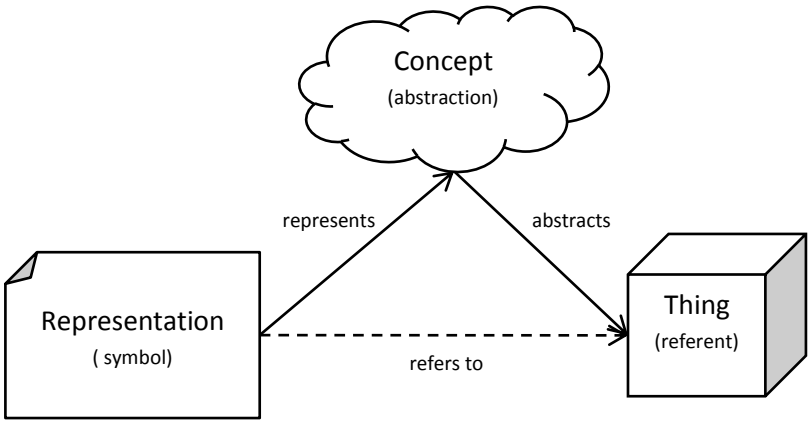



Figure 2: Triangle of reference (terminology adapted to conceptual modeling)

To illustrate this definition, consider the statement “There is a person called John, who owns a red car”. This is a conceptual model of a situation (according to our definition), since it is a *representation* of concepts, in the form of a sentence in the English language. This sentence represents the *concepts* of John, people, cars, the color red, and their relations. These concepts are mental structures, which exist in the minds of people, such as the reader and writer of the statement that represents this conceptualization. The conceptualization is an abstraction of reality itself, focusing on those aspects that are deemed relevant by the creator of the conceptualization. In this case, the name of the person that owns the car is included in the conceptualization, but not his age, physical location, etc.

What separates conceptual models from sentences in natural language is that conceptual models are meant to be *formal* descriptions, i.e., conceptual models are intended to have well-defined semantics. In Computer Science, this generally means that their semantics are defined in a mathematical or logical structure such as first-order logic. This definition is called the *formalization* or *axiomatization* of the conceptual modeling language. While the intent of the formalization is to provide semantics, the use of a formalization also allows automation to be applied to the conceptual model (see Section 2.4.1).

However, such formalizations are not the primary representation of conceptual models. Conceptual models are intended to be read by humans, so that humans can communicate about the aspects of reality that are described. This has consequences for both the conceptualization and the representation. Both should be constructed with understandability by humans as a foremost concern.

## 2.2 Conceptualizations and Ontologies

In Computer science, conceptual models of domains are traditionally called ontologies [4], which are related to the philosophical field of Ontology, the study of *what exists*. In Philosophy, the term 'ontology' is generally used to refer to a classification or description of domain-independent aspects of entities (e.g., the nature of identity, types, properties, part-whole relations, and so on). In Computer Science, 'ontology' often refers to a classification or description of what exists in the particular domain that is being modeled. To distinguish these, we call the former *foundational ontologies,* and the latter *domain ontologies*.

The relation between ontologies and individual entities in a domain can be expressed as follows: both foundational and domain ontologies define a set of world states that are valid according to the ontology. A world state refers to a configuration of individual entities in the domain (see Figure 3).

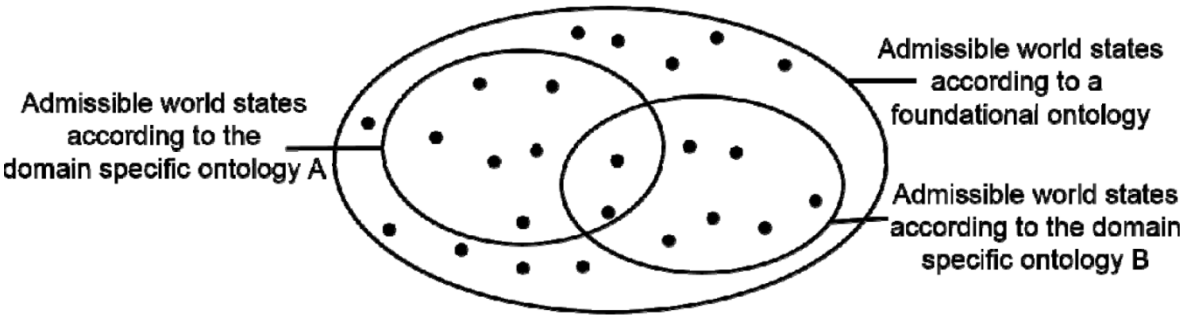



Figure 3: Sets of admissible world states according to different ontologies [11]

In [3,4], the authors argue that in order to achieve its goals of being a suitable and clear representation of a domain, a domain ontology should be based on a foundational ontology. For example, consider a domain ontology that describes cars, people, roads, cities, drivers, passengers, and their relations. A world state in which John is driving a red Volkswagen Beetle from Enschede to Amsterdam is admissible according to this ontology. This domain ontology should be based on domain-independent concepts, such as kinds and roles. A foundational ontology captures the domain-independent semantics of these concepts, such as the notion that kinds can play roles, but properties cannot. By basing the domain ontology on a foundational ontology, it is possible to create a domain ontology that states that persons can play the role of driver with respect to a car, but it is not possible to create a domain ontology that states the color of a car is playing a role with respect to anything. In this way, the grounding of domain ontologies in a foundational ontology prevents the expression of conceptualizations that are incoherent according to the foundational ontology.

Figure 4 summarizes the different types of conceptualization and representation and their relations.

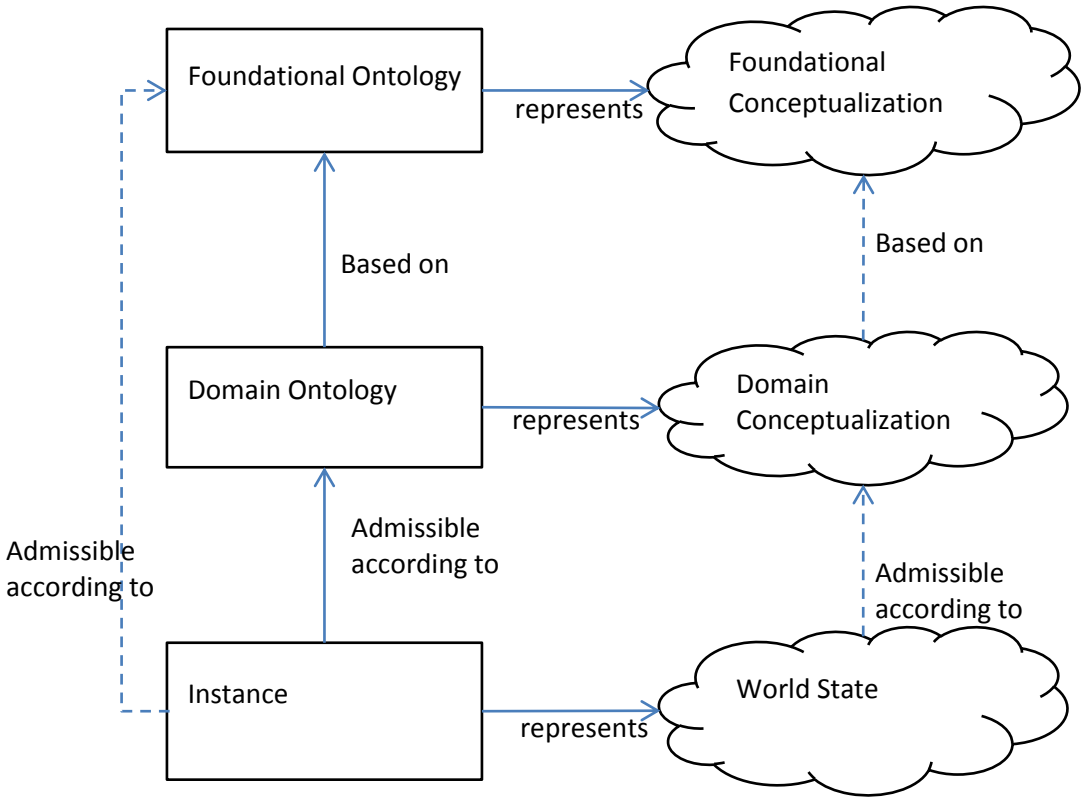



Figure 4: Relation between ontology and conceptualization

## 2.3 Quality of conceptual models

A conceptual model *refers to* reality (see Figure 2), and the purpose of the model is to do so *accurately*. This reference occurs indirectly: aspects of reality are mapped to concepts, which are mapped to a representation. The quality of a conceptual model therefore depends on the faithfulness of these two mappings. The degree to which a representation matches a conceptualization is named *comprehensibility* or *clarity*, and the degree to which the conceptualization is a suitable abstraction of reality is called *domain appropriateness* [10].

Other concerns may be important for certain conceptual modeling languages (see Section 2.4), but comprehensibility and appropriateness are the most essential, because they are directly derived from the defined purpose of conceptual models.

### 2.3.1 Clarity

In [4], Guizzardi argues that comprehensibility is based on the degree of *isomorphism* between concept and representation. Ideal comprehensibility is reached when there is a one-to-one mapping between elements in a representation and elements in a model. The definition of isomorphism can be split into four properties (injectivity and surjectivity in both directions), which Guizzardi terms *lucidity*, *soundness*, *laconicity*, and *completeness*. He shows that a mismatch between model and

concept reduces the quality of the model, and that a mismatch between domain conceptualization and domain model reduces the quality of the domain model, impacting the models created from it.

In practice, representations of conceptual models often take the form of diagram-based languages [4]. The use of visual elements impacts the clarity of the representation [8,10]: The use of visual aspects can suggest commonalities and differences between represented elements, and it can emphasize or de-emphasize aspects of the conceptualization. We refer to Chapter 6 for a detailed discussion of how visual aspects of a representation affect clarity.

### 2.3.2 Domain appropriateness

The domain appropriateness of a conceptual model is the degree to which its conceptualization suitably represents reality. At first glance, this is the responsibility of the creator(s)of a conceptual model: the skill of conceptual modeling lies in the creation of *suitable* models. However, the design of a conceptual modeling language should aim to facilitate the modeler in achieving his goals. As stated in Section 2.2, the suitability of domain-specific conceptualizations is dependent on the suitability of the foundational ontology that underlies these conceptualizations. A conceptual modeling language can therefore improve the domain appropriateness of conceptual models by the use of a suitable foundational ontology.

The suitability of a foundational ontology for conceptual modeling depends on how well it meets the purpose of conceptual modeling, i.e., the usefulness of its (domain-independent) abstractions with respect to understanding and communication between people about real-world phenomena. A foundational ontology that is appropriate for conceptual modeling should therefore focus on a common-sense interpretation of reality, at the level at which people experience it. For instance, it should be capable of expressing mental constructs (e.g., appointments, ownership, marriage) as well as physical entities (e.g., people, buildings, products).

## 2.4 Other aspects

The design of conceptual modeling language is not only aimed at maximizing the quality of the produced conceptual models. Often, a language has other specific goals, which require a trade-off between the accuracy and appropriateness of the produced conceptual models and these other factors. We discuss the most prevalent of these factors and their impact on the quality of the conceptual models.

### 2.4.1 Formalization

Conceptual modeling languages use a mathematical system, such as first-order logic, to capture the semantics of their conceptual models. This is referred to as the formalization of the model. A formalization is an unambiguous representation of the semantics of the model. This improves its usefulness as a tool for communication between humans, but it has also the advantage of making the semantics suitable for automation, such as syntactic validation, visualization and simulation, or even theorem proving or other automated reasoning tasks.

More expressive models and more elaborate automation tasks both require a stronger formalization, and this forces a trade-off between the expressivity of the conceptual modeling language, the com-

putational complexity of automated reasoning tasks, and the performance and tractability of these tasks.

OntoUML favors language expressivity over performance. Other conceptual modeling languages, such as Z [12] and the Web Ontology Language [13], aim at enabling automated reasoning, and have been designed with a simple formalization as an explicit criterion.

### 2.4.2 Implementation domain concerns

Some modeling languages are created with explicit implementation concerns. ER diagrams [14], for example, are intended to be used as a conceptual modeling language, but with the express purpose of modeling information systems and translating the model to a database design. This concern implies that ER diagrams favor the performance of an implementation over the accuracy of the representation.

### 2.4.3 User base

Some modeling languages have originally been developed for a different purpose, and have been co-opted for use as a conceptual modeling language. Generally, the constructs of the modeling language are related to the domain for which the language was originally intended, rather than the purpose of expressing real-world structures and phenomena. The most notable example is UML [5], which was originally developed to be a software design language.

Because the use of UML is ubiquitous, several studies have focused on analyzing its suitability as a conceptual modeling language, and on modifying it towards this end. Among such adaptations of UML is OntoUML [4], the language on which this work is based.

# 3 Unified Foundational Ontology

This chapter presents the Unified Foundational Ontology (UFO) [4]. UFO is a foundational ontology, which is a conceptualization of the domain-independent aspects of reality. It is based on theories from Philosophy, Cognitive Science, Linguistics and Knowledge Representation, and it is designed for the purpose of conceptual modeling. It has been used to evaluate and redesign existing conceptual modeling languages, as well as to provide real-world semantics for their modeling constructs [15]. In [2], UFO has been used to redesign UML to make it more suitable for conceptual modeling, a process that resulted in OntoUML.

A basic understanding of UFO and the terminology that is used is required for understanding conceptual models created with OntoUML. We outline the global structure of UFO as presented in [4]. Here we aim for a general understanding and therefore we omit some details. We illustrate the concepts with examples where necessary.

Section 3.1 explains the core concepts of UFO. Section 3.2 deals with the classification of substances, Section 3.3 with properties, and Section 3.4 with relations. Section 3.5 discusses UFO's implementation in OntoUML.

## 3.1 UFO core

The Unified Foundational Ontology is a classification of the elements that make up reality into different categories. In this section, we briefly describe the principal categories of UFO, from most general to more specific. A further subdivision of the fundamental categories is found in the subsequent sections.

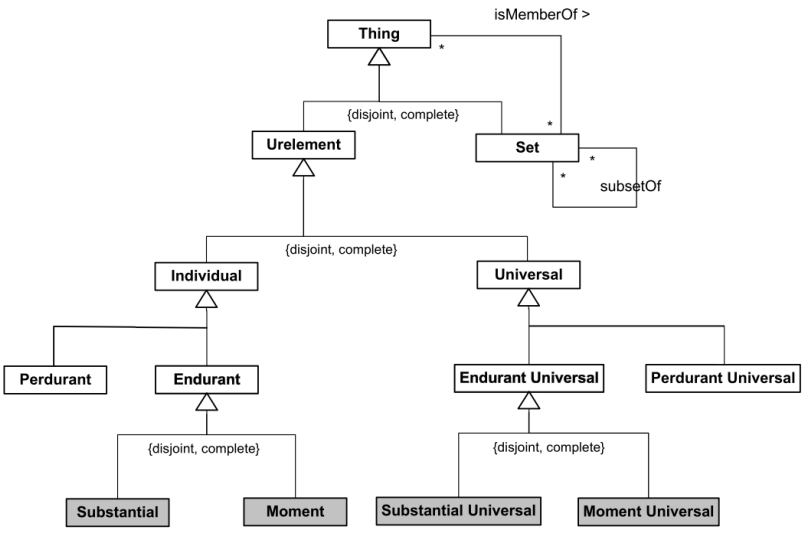



Figure 5: Fundamental categories in UFO

The topmost category is that of *things*, and this category encompasses all things that can exist. Things are divided into *sets* and *urelements,* which are things that cannot be considered as sets.

Urelements are divided into *universals* and *individuals*. A universal is an entity that can in some way be wholly and repeatedly present at many different times and places. It is a *pattern* of reality that

can be *instantiated* multiple times. Such instances are called *individuals*, and these are always single contiguous entities that do not recur. For example, 'Person' is a universal, and 'a particular person called John' is an individual. When the individual person 'John' no longer exists, he can never exist again, but the universal 'Person' can be instantiated as another Person, and is wholly present in every other person.

A universal is not simply a set of its instantiating individuals. Two universals that have the same extension (set of instantiations) are not identical. Universals are instead identified by supplied principles of application and identity, and/or in terms of the causal powers bestowed by them [16]. For example, in a world where the universals 'cars' and 'blue things' have the same extension (the only blue things are cars and vice versa), 'blue' and 'car' are still different, because 'blue' is merely a *property* of a thing, whereas 'car' describes the identity of a thing, as well as bestowing causal powers (cars can move).

Universals and individuals can be classified as *endurants* or *perdurants*, based on their behavior with respect to time. Informally, endurants *exist in time*. They are wholly and completely present at every moment. For example, a person observed at two different moments is still identified as the same person, and thus is an endurant. Perdurants *happen in time*, and are identified by the sum of their parts that exist in different moments. 'A conversation', 'a cooking session' and 'a transaction' are all perdurants.

OntoUML is a conceptual modeling language concerned primarily with modeling the *structure* of domains. As such, it builds on the ontology of *endurants* provided by UFO. For this reason, we omit a discussion of perdurants.

UFO further divides endurants (both individual and universal) into substances and moments.

Endurants which cannot exist independently of others *and* inhere in other endurants are called *Moments*. Inherence is a specific type of existential dependence, difficult to characterize precisely. Informally, when A inheres in B, A is a property or attribute of B, or A is a way of being of B. Moments cannot be conceived independently of the particulars they inhere in. For example, 'John's height' inheres in 'John'.

Endurants that do not inhere in others are called *substantials.* These are *things*, such as 'cars' and 'people'. While they cannot inhere in other endurants, substantials may still be existentially dependent on others. A person, for example, depends existentially on their brain, but 'person' is not a moment because a person does not *inhere in* their brain.

## 3.2 Substantials

UFO defines an elaborate subdivision of substantial universals into categories such as *roles*, *mixins*, *collectives*, *kinds* and *subkinds*. This subdivision is based on metaproperties (properties of universals) as defined in [6]. We discuss these principles informally and then explain how OntoUML uses them to classify the universals.

### 3.2.1 Identity

The *identity* of a universal is defined with respect to an *identity condition*, a relationship defined between two individuals. If it holds, the two individuals are the same. Universals either *carry* an identity condition (inherit it from a generalization) or *supply* it. Each entity has only one identity condition.

'Person' is a universal that supplies an identity condition. 'Man' and 'woman', as types of person, carry the identity, but do not supply their own.

### 3.2.2 Rigidity

*Rigidity* is a metaproperty, as described in [6]. In OntoUML, rigidity is a property of universals. A universal is *rigid* if it is essential to all its instances. A universal is *nonrigid* if it is essential to some but not all instances. A universal is *antirigid* if it is not essential to any instance. A universal is essential to an individual if and only if whenever the individual exists, it is an instance of that universal.

'Person' is an example of a rigid universal. Persons are always persons, and an entity cannot cease to be a person without ceasing to exist altogether. 'Student' is an antirigid universal. Every student instance is not *essentially* a student, i.e., all students can start or stop being a 'student' at some point, while still continuing to exist. 'Things you can sit on' is an example of a *nonrigid* universal. It is essential to individual 'chairs': if you can't sit in a chair, it is no longer a chair. Crates or suitcases can also be used for sitting on, but this is not an *essential* property of these things.

### 3.2.3 Substantial universals

Figure 3 shows the classification of substantial universals as defined by UFO, according to the principles of identity and rigidity. Instances of these categories adhere to additional rules specifying how they may relate to other universals.

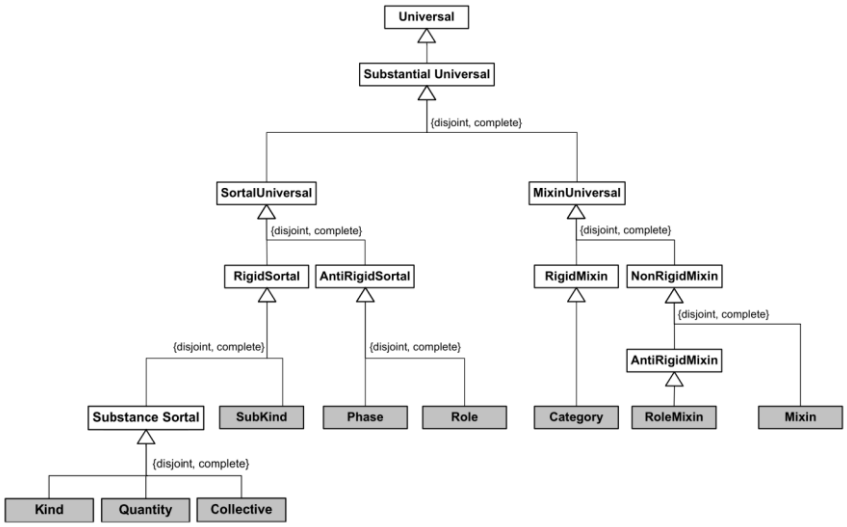



**Figure 6: Substances in UFO. The categories used in OntoUML are marked in grey.**

*Sortal universals* have an identity condition, while *mixin universals* do not. *Substance sortals* supply the identity condition, and the other sortals carry the identity condition. Substance Sortals are all rigid. There are three types of substance sortals: Kinds, Quantities and Collectives. Kinds are sub-

stance sortals that are whole and singular, such as person, car, and planet. Quantities are nominalizations of amounts of matter like water, sand, sugar, martini and wine. Their identity does notchange when adding or removing some amount. Collectives are representations of collections in general, such as a forest, a deck of cards, a pile of bricks and a pack of wolves. Subkinds are types of Sortal that specialize other rigid sortals, e.g., man and woman, truck, van and sports car.

Phases and Roles, which are the two types of antirigid sortals defined in UFO, represent temporary aspects of the sortals they specialize. *Phases* represent possible stages in the history of a substance sortal. For example, 'alive' and 'deceased' are possible stages of a Person. Phases form a partition of the substance sortal they specialize, based on intrinsic properties of that sortal. *Roles* represent temporary aspects of rigid sortals based on properties that relate the sortal to external entities. For example, Student is a role played by a person that relates them to another entity, such as a University.

Mixins are the bearers of properties that apply to dispersive types, i.e., types that are not related by a grouping principle of identity. *Categories* are rigid types that represent essential properties that are common to all their instances. An example is 'Legal Entity', which may apply to both persons and organizations. *Role Mixins* represent abstractions of common properties of roles. *Mixins* represent properties that are essential to some of its instances and accidental to others. For example, 'thing you can sit on' is essential to chairs and accidental to crates.

## 3.3 Moments

*Moments* are endurants that inhere in other endurants. They are the endurants that are 'about' other endurants, which in some way *describe* the endurant they inhere in.

Using several key distinctions, UFO classifies moments into categories (Figure 7). Moments that inhere in one endurant are *intrinsic moments*. Intrinsic moments are further divided into *qualities* and *modes. Relators* are a type of moment that inheres in more than one endurant.

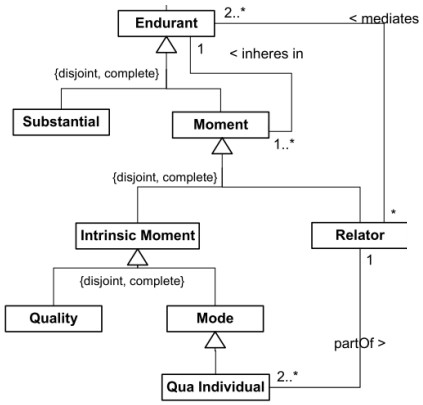



Figure 7: Moments in UFO

The simplest type of intrinsic moment is *quality*. A quality is an attribute of an endurant that is measurable in some way. More precisely, a quality is an individual intrinsic moment inhering in an individual endurant that describes that endurant by relating it to a *quale* on a *quality structure*. For example, 'the weight of John' is a quality inhering in 'John' that relates John to the quale '75kg' on the quality dimension 'weight in kg'.

There are two kinds of quality structures: the *quality dimension* and the *quality domain*. A quality dimension is a set of values that represent a measurement of a single quality (e.g., the positive real numbers for a temperature scale, kg for weight, a 1-10 value for bitterness). Two quality dimensions are called *integral* when one cannot assign a value to a quality on one dimension without assigning it to the other (e.g., hue, saturation, and brightness, when considering color). Otherwise, two dimensions are *separable*. A group of integral quality dimensions that are separable from all other dimensions form a *quality domain*. For example, HSB is a quality domain for Color. A particular value on a quality structure is called a *quale*.

In the realm of individuals, we speak of an individual endurant bearing a quality that is represented by a quale. In the realm of universals, we speak of an endurant universal characterized by a quality universal that is associated with a quality structure. An example of these relations is shown in Figure 8.

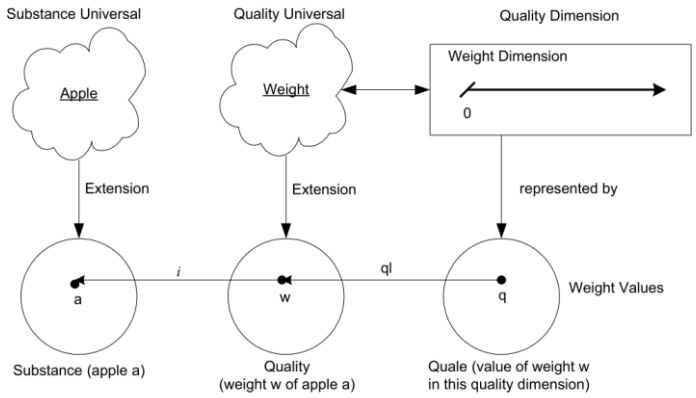



**Figure 8: Example of a quality**

There can be multiple quality structures used to represent a quality. The quality of Color, for example, can be represented using a wide variety of quality domains, such as the RGB, CYMK, or CIE 1931 color spaces, or even a set of colors such as 'red', 'orange', 'yellow', etc.

*Modes* are another type of intrinsic moment. Modes cannot be described by a single quality structure, but can be conceptualized by multiple separable quality structures. Examples include beliefs, desires, intentions, perceptions, symptoms and skills. Modes can bear other intrinsic moments, so they may be described by a group of separate qualities.

*Relators* are individuals with the power of connecting entities. For example, a flight connection is a relator that connects airports, and an enrollment is a relator that connects a student with an educational institution. A relator is a non-intrinsic moment, and it is composed of the *qua individuals* that inhere in the individuals it mediates. A *Qua Individual* is a particular type of mode that, intuitively

speaking, describes the way an object participates in a certain relation. Relators are a key element in defining material relations.

For example, the substances 'John' and 'The University' are mediated by the relator 'the enrollment of John at The University'. The enrollment relator is composed of the qua individuals 'John as a student of The University', which inheres in John, and the qua individual 'the university as a school for John', inhering in The University.

## 3.4 Relations

Relations are entities that glue together other entities. Each relation has a number of relata as arguments, which are connected or related by that relation. We make a fundamental distinction between formal and material relations.

*Formal relations* hold directly between two entities without any further intervening individual. For example, the formal relation 'John *is taller than* Mary' holds between John and Mary without any further individuals. There are two types of formal relations, namely *comparative* and *basic* relations.

*Basic formal relations* form the mathematical framework of UFO itself. They are domain-independent, and include concepts such as specialization, inherence, instantiation and association. These are not analyzable within UFO itself.

*Comparative formal relations* are domain-specific relations between entities that are founded in qualities that are *intrinsic* to their relata, and so they can be reduced to relations between these qualities. 'John is taller than Mary' depends on the length of both individuals, and knowing their length is all that is required to establish the relation.

*Material relations* depend on some external entity. A material relation is *derived from* a relator, and its relata are mediated by that relator. Examples are employments, kisses, enrollments, flight connections and commitments.

*Meronymic relations* are relations between parts and whole. UFO's classification of the part-whole relations distinguishes between *essential*, *mandatory* and other types of parthood. Because parthood relations are not relevant for this work, we omit further detail.

## 3.5 OntoUML

UFO was initially applied as a framework for evaluating the ontological foundations of UML. This resulted in the definition of OntoUML, which a UML re-design, implemented as an UML Profile that extends the UML classification structure with the concepts of UFO. This provides ontological foundations to UML, making it suitable to be used as a conceptual modeling language. OntoUML can be used for the definition of domain ontologies.

The mapping of UFO's ontological concepts to the model elements of OntoUML is straightforward in most cases. The categories of UFO are present in OntoUML as specializations of existing UML model elements (see Figure 9). Instances of these categories (Universals) may be related by specialization, as described in previous sections. There are a number of restrictions that apply to these specializa-

tion relations (e.g., a kind may not specialize another kind, a phase must specialize a sortal), and these are included in OntoUML.

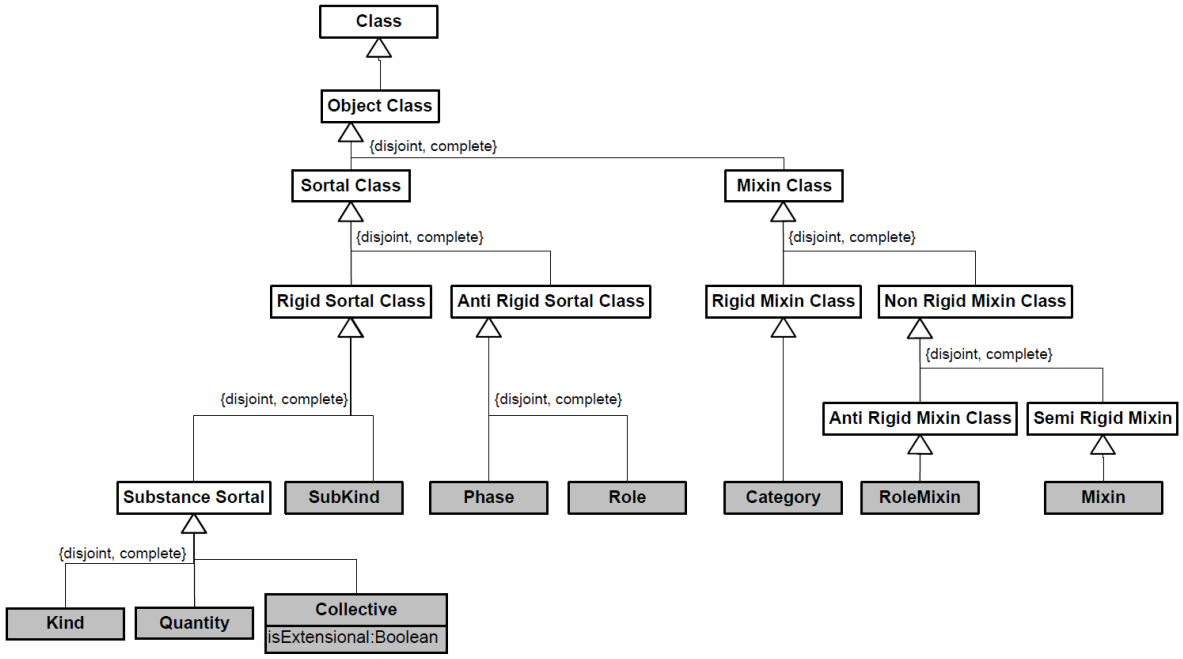



Figure 9: A fragment of the representation of UFO substances in OntoUML, as specializations of the UML Class [4]

Figure 10 shows an example domain model described with OntoUML: a family tree. The model contains one kind, *person*. A person can be either a *man* or a *woman*, modeled as subkinds. The age of a person is partitioned into three phases. Finally, persons can be either *alive* or *deceased.* Persons are related to other persons through *conception* relators. With respect to these relators, they can play the role of *father*, *mother*, or *offspring*. Elements of this domain ontology are used as examples throughout the remainder of this work.

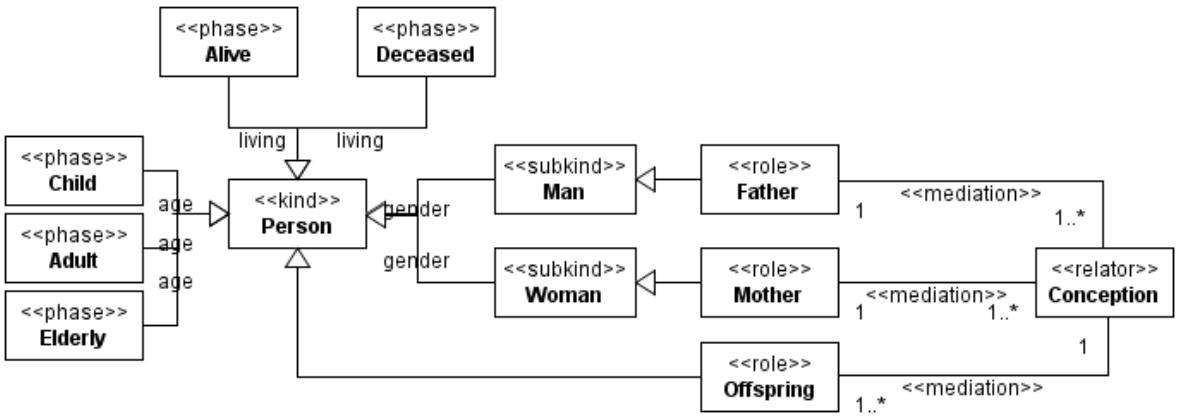



Figure 10: OntoUML model for a family tree

# 4 Simulation of OntoUML models

OntoUML is a conceptual modeling language based on UFO, and it is supported by a toolset that provides a graphical editor for the specification of domain ontologies. In order to allow the modeler to validate their models, a transformation to a formal specification language was developed [7]. Conceptual models can be translated to a specification in Alloy [17], which is a modeling language based on first-order logic. This specification can be simulated using the Alloy Analyzer, which is a tool that generates example instances that conform to a given Alloy specification.

Using this approach helps the modeler understand or validate the OntoUML conceptual models. However, by simulating OntoUML domain models *in time*, the model simulator characterizes OntoUML as a language capable of expressing time-related aspects of a domain, something OntoUML was not designed to do. The goal of this work is to supplement OntoUML in precisely this area. This chapter examines the model simulator, and the capabilities of OntoUML with respect to expressing dynamic aspects of a domain ontology.

Section 4.1 gives an overview of Alloy and the Alloy analyzer. Section 4.2 describes the structure of the transformation from an OntoUML model to an Alloy specification. Section 4.3 details the temporal structure defined as part of the Alloy specification. Section 4.4 examines dynamic aspects of domain ontologies and the expressiveness of OntoUML in this regard. Section 4.5 summarizes our findings.

## 4.1 Alloy

Alloy is a specification language based on many-sorted first-order logic, which can be used to create constraint-based specifications of structures. An Alloy specification is a series of declarations that define types, relations, predicates and facts, which together constrain a space of admissible instances.

Specifications can be checked and simulated using the Alloy Analyzer, which generates admissible instances from a specification. It does this by transforming the specification to a Boolean formula, and using a SAT-solver to generate a configuration that satisfies the formula. This configuration is then translated back to an instance model that conforms to the specified constraints.

Satisfiability in many-sorted first-order logic is undecidable. Alloy resolves this by limiting the search for conformant models to a predetermined ‘scope’, which is an upper bound on the number of each type of object contained in the model. The validity of assertions is determined by generating counterexamples, and therefore the Alloy Analyzer can only determine the validity with respect to this limited scope. I.e., it cannot prove that there is no larger scope in which a counterexample does exist. Furthermore, even in a limited scope, the satisfiability problem is NP-complete, and it is impractical to simulate complex specifications within large scopes. Alloy works on the principle that most errors in models can be found with small scopes. It aims to be a practical tool used as an aid for modelers, a lightweight alternative compared to full-fledged model checkers and theorem provers.

Using Alloy to simulate OntoUML conceptual models provides the following benefits:

1. Transforming a conceptual model to an Alloy specification provides an explicit formal semantics, as the Alloy specification is a precise definition of admissible states of affairs in the domain. Such a formalization already exists for OntoUML (that's what the transformation is based on) but the transformation automates the application to domain ontologies.
2. The Alloy Analyzer can be used for simulation. By generating examples that conform to the domain model and examining them, a modeler can verify heuristically that the domain model expresses his intentions.
3. The modeler can explicitly verify intended properties of the domain model by specifying his intentions in Alloy as assertions and using the Analyzer to find counterexamples.

## 4.2 Transformation of OntoUML to Alloy

In [7], a transformation from OntoUML domain models to Alloy specifications is defined. This transformation consists of three elements: foundational ontological properties; a temporal structure; and the translation from OntoUML categories to Alloy formulas.

OntoUML is based on certain fundamental ontological properties, such as rigidity (see Section 3.2.2), existence, necessity, and essential-ness in part-whole relations. Some of these properties deal with alethic modality or the notion of what is *possible*. Rigidity is the prime example. Informally, rigidity can be described as the *possibility* of change: Person, a Kind, is a rigid entity, because individual persons cannot stop being persons and continue to exist. Student, a Role, is anti-rigid, because it is possible for an entity to stop being a Student while continuing to exist. The semantics of these properties in terms of Alloy formulas are defined in a separate specification, using S5 modal logic.

Because Alloy does not natively support modal logic, a second specification is added that defines a structure of multiple worlds as a foundation. The modal properties of domain entities can then be defined in terms of these multiple worlds: *necessary* properties exist in all worlds, and *possible* properties exist in some worlds.

In the Alloy specification, the multiple worlds used by the definition are ordered as a temporal structure, with multiple worlds representing past, future, and possible worlds. This is not a requirement of modal logic, but an interpretation of the multiple worlds in a fashion that is intuitively understandable by users. The details of this structure are discussed Section 4.3.

Lastly, the transformation defines how each of the constructs of a conceptual model should be translated to declarations in an Alloy specification. Only the domain-specific elements of an OntoUML conceptual model are translated to the Alloy specification, and there is no explicit mention of OntoUML's categories in Alloy (e.g., kinds, modes, relators), beyond a broad separation into *objects* and *properties*, which correspond to OntoUML's substances and moments, respectively.

Figure 11 provides an overview of the transformation infrastructure.

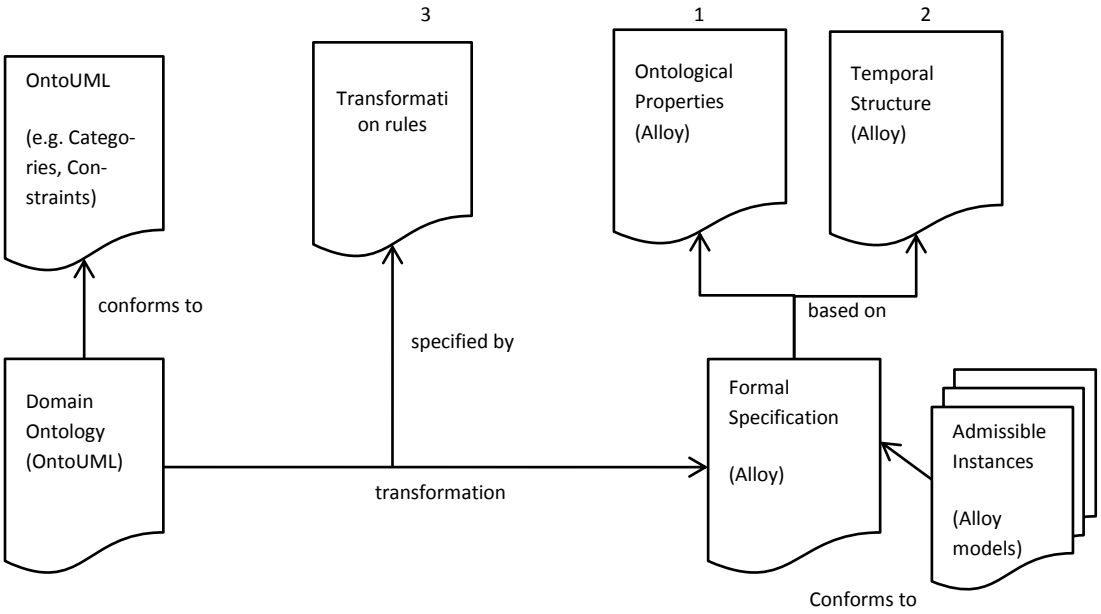



**Figure 11: Transformation of an OntoUML model to an Alloy specification**

## 4.3 Temporal structure

In order to allow validation, the simulation of OntoUML models must demonstrate the properties of its elements. Some of these properties (specifically rigidity) deal with alethic modality, i.e. the notions of possibility and necessity. These cannot be expressed by a simple structural simulation. For example, *possibility* can be expressed by showing a world in which the *possible* structure occurs, and another in which the structure does not occur.

Conventionally, an instance of a domain model is conceived as a description of the state of a domain at a single moment in time. For example, UML Class Diagrams are instantiated by UML Object Diagrams, which describe the structure of the domain at a specific moment. In order to validate OntoUML's modal properties, the instances of OntoUML domain models must be able to contain multiple worlds. Because the generated instances are also intended to be simulations that may be inspected by people, a world structure is used that can be interpreted temporally (i.e., the accessibility relation between worlds is interpreted as one of succession in time). We call an instance of this structure a *history*.

The world structure that is used is based on actualism [7]: there is one actual *current* world. In addition, there may be a single path of *past* worlds that occurred previously. From these past worlds *counterfactual* worlds may branch, which are worlds that could have possibly occurred, but did not. The current world may be succeeded by branches of *future* worlds, which are worlds that may possibly occur after the current world (see Figure 12).

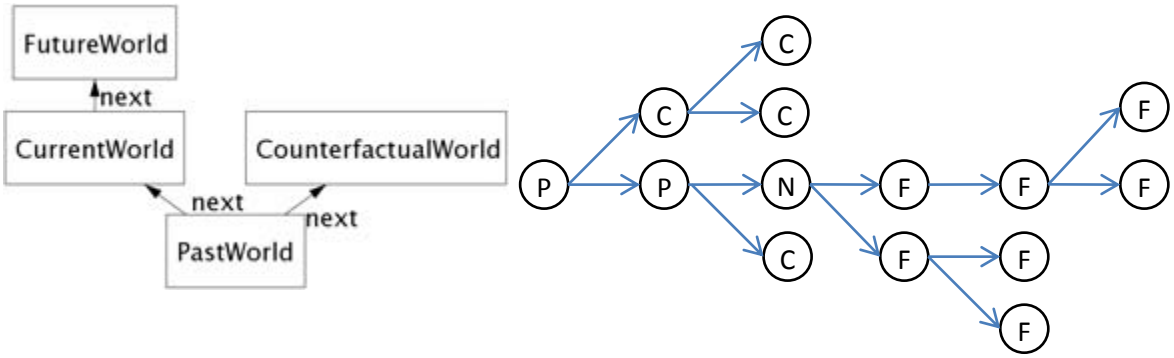



Figure 12: The temporal ordering of worlds [7] (left), with an example instance (right)

Each of these worlds contains individuals conforming to an OntoUML model. The succession of worlds, with individuals changing over time, provides the demonstration of OntoUML's modal properties (see Figure 13).

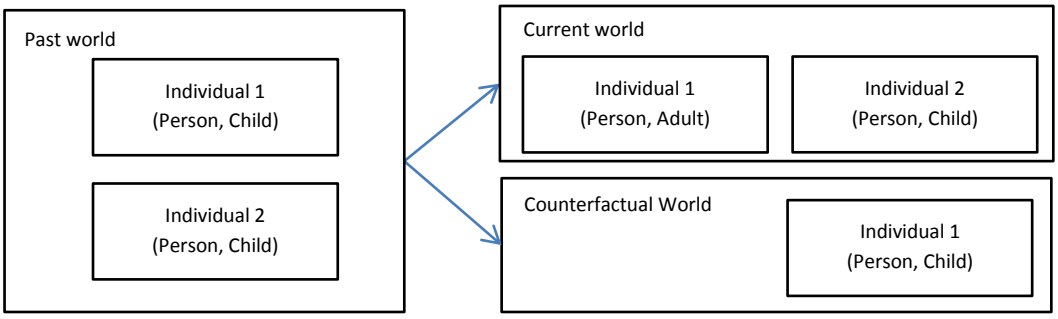



Figure 13: Example history, demonstrating the antirigidity of phases.

## 4.4 Dynamic Aspects and OntoUML

The histories generated by the model simulator can be interpreted as the instances of OntoUML domain ontologies. A domain ontology can be described as defining a set of admissible world states (see Section 2.2). Because the Alloy model simulator defines histories instead of states of a single world, this definition needs to be amended, so that worlds are replaced with histories. OntoUML, combined with the temporal structure defined by the transformation to Alloy, comprises a foundational ontology that defines a set of admissible histories. Domain ontologies created with OntoUML constrain this set of histories to those that are admissible according to the domain conceptualization.

However, OntoUML was developed as a language for structural models. This means it has been designed with the purpose of constraining the static aspects of a domain. With respect to our definition of ontology, this means that OntoUML is aimed at constraining admissible world states.

Complementary to the static aspects of a domain, we consider the existence of dynamic aspects of a domain, defined as those aspects that can only be expressed in terms of more than a single world. These dynamic aspects should not only be considered in terms of OntoUML, but in terms of domain conceptualizations. Many domains contain time-related concepts, and if OntoUML instances have a time-dimension, it is a matter of interest to see how well OntoUML can express these concepts.

We discern three categories of dynamic aspects: *temporal entities*, *perdurants*, and *behavior of endurants*. The following sections discuss each of these categories,their treatment in OntoUML and its representation in Alloy.

### 4.4.1 Temporal entities

Fundamental to the notion of *dynamic aspect* is the structure of time in which these aspects occur. In order to be able to describe dynamic aspects, a conceptualization of *time* is required.

There are many possible conceptualizations of time. Time can be expressed in numerical terms, using timestamps. Time can also be expressed using intervals and the relations between them, describing situations that overlap, occur consecutively, or are disjoint. Examples are the category of Chronoids in the ontology GOL [18], and Allen's interval algebra [19].

The notion of time used by OntoUML/Alloy is described in Section 4.3. Additional constraints exist, which enforce that "all entities must be continuous in time". A person cannot exist today, then disappear tomorrow, then exist again next week. OntoUML/Alloy's definition of histories constrains the set of admissible histories: a history with two Present Worlds is inadmissible, as are cycles, or a future world preceding a past world.

The structure of histories forms the basis of the definition of other dynamic aspects. In OntoUML/Alloy, these constraints exist at the level of the modeling language, and a modeler of domain ontologies cannot directly interact with this temporal structure.

### 4.4.2 Perdurants

Perdurants are discussed in Chapter 3: they are entities that *happen* in time. They cannot be fully described by a snapshot of a single moment. Examples of perdurants are events, processes, and actions.

Foundational Ontologies like UFO-B and GFO classify entities of this kind. OntoUML contains no explicit account of perdurants, so it is not possible to directly represent these in a domain ontology with OntoUML. However, OntoUML's relators can be interpreted as an indirect representation of perdurants. For example, a relator that connects a student to a university can be interpreted as a representation of the process of that student's studies at that university. Alternatively, the creation of a relator can be interpreted as being caused by an event: an "ownership" relation between a person and a car may be caused by a sale event.

However, such interpretations are imprecise, as they make no distinction between types of perdurants. More complex perdurants (processes with multiple steps, complex chains of events) cannot easily be described by relators. Relators are insufficient as a representation of perdurants.

### 4.4.3 Behavior of endurants

Behavior specification defines the constraints on the change in existence/relations between endurants in time. Behavior is related to perdurants, in the sense that behavior can be the effect of a perdurant (e.g., the process of walking changes the location of a person).

Behavior is a broad category and can be modeled in many ways. At the level of foundational ontology, ontologies like BWW model behavior in terms of laws. Transformation laws constrain how an object may change over time. An alternative model is that of UFO-B, which gives a dispositional account of behavior: endurants have a disposition towards certain events, and events then bring about situations.

To some extent, it *is* possible to model behavior with OntoUML, as there are some dynamic aspects inherent in the categories of OntoUML that constitute behavior. The definition of those categories constrains admissible changes to individuals that are their instances. For example, consider an ontology that defines two Kinds: Chairs and Persons. As kinds are a rigid substantial, an individual can be of only one Kind in its history.

- A World in which there is an individual X that is a Chair is admissible;
- A World in which there is an individual X that is a Person is admissible;
- A History that contains both these worlds consecutively, so that X is first a Chair and then a Person, is not admissible.

In a similar vein, Relators cannot have different relata in different worlds, and relation ends marked as essential are similarly constrained.

There are many other types of behavior, specifically those that do not arise from the *category* of an endurant, but those that are related to the specific universal. For example, consider the category of phases. Phases are grouped into phase partitions, and a single phase from the partition manifests itself at any given world. When considering a specific universal, such as the kind person, with phase partition child/adult/elderly, additional behavior exists in the *ordering* of these phases.

Persons should be in the ‘child’ phase when they begin to exist, and then possibly progress to ‘adult’ and then ‘elderly’. These restrictions cannot be expressed in an OntoUML domain model.

A modeler *can* make statements about this type of behavior directly in the Alloy representation of a domain model. This is a manipulation of the set of admissible histories, directly in the formalization. It does not affect the domain ontology itself, it merely corrects the disparity between the intended histories and the admissible histories, in a way that is not possible in an OntoUML definition.

## 4.5 Final Considerations

The simulation of domain models with Alloy provides a powerful tool to aid in the creation of domain ontologies with OntoUML.

In order to demonstrate the alethic modal properties of OntoUML’s categories, the Alloy-based model simulator goes beyond a mere structural example of a situation in a domain, and instead uses *histories* with multiple worlds as instances of OntoUML domain ontologies. This representation of

models *in time* creates expectations about the capabilities of OntoUML to express time-related aspects of a domain, something OntoUML was not designed to do.

To align the expectations created by the temporal structure of model instances with the expressiveness of the modeling language requires an extension to OntoUML. Specifically, what is missing is the expressivity to describe behaviors related to universals of a particular domain, rather than behaviors that are derived from the categories of domain entities.

# 5 OntoUML Behavior Specification Language

To complement OntoUML, a technique to model the behavior of domain entities is required. This thesis proposes a modeling language for this purpose, which we call the OntoUML Behavior Specification Language (OBSL).

This chapter describes OBSL. Section 5.1 describes the notion of behavior as modeled by OBSL. Section 5.2 describes how OBSL models the notion of 'circumstances' under which particular behaviors can occur. Section 5.3 describes the overarching framework of an OBSL specification. Section 5.4 describes the formal semantics of OBSL. Section 5.5 discusses the limitations of OBSL and the tradeoffs involved in its design.

## 5.1 Types of behavior

The main goal of OBSL is to be a language for the specification of behavior for OntoUML domain ontologies. This means that first and foremost, OBSL should be capable of describing behavior. Specifically, OBSL should express the domain-specific dynamic aspects at the level of the universals of a conceptual model (rather than categories, as OntoUML does). For example, a behavior specification may describe the behavior of *persons*, *students*, or *ownership*, as opposed to kinds, roles or relators. On the other hand, OBSL does not express behavior specific to *individuals*, except insofar as their individuality can be expressed in terms of the universals that classify them. For example, a behavior specification should not express behavior specifically for a person 'John', but *should* be able to express behavior for 'adult male students'.

A specification of behavior should contain two elements: an identification of the behavior, and a description of the circumstances under which the behavior occurs. We analyze these elements in the following sections.

In Section 4.4.3, we argued that behavior can be viewed as a difference in state between worlds over time. While more complex conceptualizations of behavior are possible, we consider only changes in state between consecutive worlds. 'Change in state', in this context, can be one of three things:

1. An entity does not exist in one world, but does in the next world;
2. An entity exists in one world, but not the next world;
3. An entity (a quality) changes value.

We name these behaviors *creation*, *deletion*, and *change*, respectively.

Not all behaviors are applicable to all universals. We discuss behaviors of universals systematically below:

Kinds, subkinds, and relators can have *creation* and *deletion* behaviors.

*Phases* are a special case. First, because phases are existentially dependent on their bearers, their behavior description is only applicable insofar as the bearing entity continues to exist. Second, because phases occur in partitions, the deletion of a phase must always occur simultaneously with the creation of another phase from the partition. Therefore, it is more convenient to model the behavior

of phases as *changes* between the different phases of a partition, as opposed to a pair of linked *creation*/*deletion* behaviors.

According to UFO, phases are derived purely from intrinsic qualities of the sortal that bears them (i.e., the *child/adult/elderly* phases can be derived from an *age* quality). If that were the case, behavior of phases should be specified in terms of this derivation, instead of behavioral rules. However, in practice, it is not always the case that we wish to express the derivation of a phase. For example, the precise derivation of the *alive/deceased* phases (in medical terms) is irrelevant for most models dealing with persons, even though the notion of the phases themselves *is* relevant. For this reason, we choose not to model the derivation of phases from intrinsic qualities.

The behavior of roles *is* derived from the relator that mediates them. For example, a 'father' role exists when a 'person' participates in at least one 'conception' relator as a father. The role of 'father' follows from the existence of the relator.

As qualities are existentially dependent on their bearer, creation and deletion need not be modeled. We do model their admissible transitions (in terms of values).

Characterization and Mediation relations are formed and removed with their respective qualities and relators. Their behavior does not need to be modeled separately.

Table 1 summarizes the relation between universals and allowed types of behavior.

| | Creation | Deletion | Change | Derived |
|---|---|---|---|---|
| Kind | + | + | | |
| Subkind | + | + | | |
| Role | | | | + |
| Phase Partition | | | + | |
| Relator | + | + | | |
| Quality | | | + | |
| Mediation | | | | + |
| Characterization | | | | + |
| Material | | | | + |

**Table 1: Behaviors allowed for OntoUML's universals**

OBSL's account of behavior is limited to these universals. We do not consider quantities, collectives, mixins, part-whole relations, or formal relations.

## 5.2 Circumstances

In order to specify a particular behavior, we may have to describe the circumstances under which the behavior occurs.

In the general case, *circumstances* can be interpreted as the entire history in which the behavior is to occur. Using temporal logic, it would be possible to describe complex conditions under which the behavioral rule applies. However, our aim is to provide a behavior description language that can be utilized without requiring the use of (temporal) formal logic. In order to achieve this, we limit our notion of circumstances to the pair of consecutive worlds in which the behavior occurs. This results in the circumstances being defined as (a part of) the state of the world *before* and *after* the behavior occurs.

In order to describe the circumstances under which a behavior occurs, and the changes in the world that result from that behavior, OBSL relies on the notion of a *situation*. Situations are related to the concept of *state of affairs* from philosophy, meaning roughly "a particular configuration of a part of reality" [7]. Example states of affairs are: "John visits Mary", "John has long hair", "John is a student at the University of Twente", "Mary is not taking any course."

States of affairs are used to discuss notions of possibility. A state of affairs is said to *obtain* in a world if the individuals that are present in the state of affairs are also present in the world. For example, the state of affairs "John has long hair" *obtains* in a world if John in fact has long hair in that world. State of affairs may contain relations, partial descriptions of individuals, may describe the absence of something, may be composed of other state of affairs, etc.

Unlike state of affairs, OBSL situations do not consist of individuals. Rather, they use the universals from an OntoUML domain model to describe a *pattern* that may be matched by a configuration of individuals that may occur in that domain. A Situation is said to *obtain* in a world if there are individuals that match the description in the situation. For example, the situation "A person attends a university" obtains in a world where Mary attends the University of Twente.

A situation in OBSL describes a pattern that relates individuals. The situation describes these individuals by describing their properties, in terms of the universals they instantiate or the relations that hold between them. For example, the situation "A person attends a university" relates two individuals, demanding that one is an instance of the kind *person*, the other an instance of the kind *university*, and that the material relation *attends* holds between them.

The behavior of roles and relations is not explicitly defined by OBSL (see Section 5.1). Roles and relations can be used in situations, to describe the circumstances in which behavior of another entity occurs.

OBSL is also capable of describing the *absence* of entities, either as part of the circumstances (a behavior can only occur if a certain entity is absent) or as part of the behavior itself (e.g. to express *creation,* the entity being created is shown as absent in one world and present the next).

Situations can be composed of other situations. This is used for the purpose of describing complex conditions for behavior. For example, the situation “A person that has a house but not a car” is described in OBSL as “X is a person, Y is a house, X owns Y, but it is not the case that ‘Z is a car, and X owns Z’”.

To relate individuals in different situations, situations can be parameterized with individuals. For example, “Someone is a student today, but not tomorrow” is described by two situations, “X is a student” and “X is not a student”. X is a parameter of both situations, and can be used to relate the situations by stating: “there is an X such that the situation obtains today and the second situation obtains tomorrow.”

## 5.3 Framework

The guiding principle of an OBSL specification is that "Changes in the state of the world are only possible according to the rules".

OBSL is based on a state-transition paradigm, describing the admissible behaviors for a domain entity as transitions between a ‘before’ and ‘after’ situation. These situations specify both the conditions under which the behavior can occur and the admissible changes to the world structure resulting from the behavior.

The behavior of each domain entity is defined by a set of transition rules. Each of these rules describes a particular behavior, as a before- and an after-state for the domain entity, and the conditions under which this behavior may occur, as before- and after *situations*. Together, such as set of rules constrains the behavior of instances of the domain entity: with each step to a next world, an entity must either not change, or else change in accordance with the transition rules that define its behavior.

The constraints specified in the rules of an OBSL specification are defined as an addition to the constraints derived from an OntoUML domain ontology. This means that the description of a situation may rely on rules that are not shown in the situation, but rather are implied by OntoUML (e.g., when a situation describes a substance individual in a particular phase, it is implied that individual is not in any other phase of that phase partition).

The before- and after- situations of a transition rule may *imply* behavior for entities other than the one whose behavior is specified by the rule. For example, consider a behavior rule that describes the creation of a person, and the after-situation contains a relator and role for that person (e.g., relating that person to their parents). That relator cannot exist in the world before the person was created, because it depends existentially on the roles it mediates, and the person playing the role does not exist. Therefore, the relator shows *behavior*: it is created simultaneously with the person. This behavior is not *defined* by the creation rule for persons. On the contrary, in order for this creation rule to be applicable, there must be a creation rule defined for the relator.

### 5.3.1 Transition Rules

Using the concept of situations, OBSL defines the behavior of a domain entity using transition rules. A transition rule contains two situations: a *before* and *after* situation. A transition obtains in a pair of consecutive worlds if the before-situation obtains in the first world, and the after-situation obtains in the second world. In order to relate both situations, the rule defines the individuals that parameterize them. There are different types of rules for each domain entity.

For example, consider the behavior "Persons can change from children to adults." This behavior is implemented as a transition rule containing the before-situation, in which a person is a child, and an after-situation, in which that person is an adult.

In Section 5.1, the types of behavior exhibited by each category of entity are summarized. A type of transition rule is defined for each of these behaviors.

For example, the example transition rule above is a *phase change rule* for the *age* phase partition. It necessarily includes an individual X, an instance of one phase in the before-situation and an instance of another phase in the after-situation, in which both phases are members of the *age* phase partition.

### 5.3.2 Rulesets

The behavior of domain entities of an OntoUML model is described by rulesets, composed of a group of rules that is applicable to that domain entity. An OBSL specification contains one ruleset for each Kind, Subkind, Phase partition, Relator, and Quality that is present in an OntoUML model.

A ruleset governs the behavior of their respective entities, by specifying that for each instance of that entity, one of the transition rules obtains in each pair of consecutive worlds, or no behavior occurs. In other words, in each world, an individual may exhibit one of the applicable behaviors as defined by its ruleset, or it may transition into the next world without changing.

For example, consider the *age* phase partition, comprised of the phases *child*, *adult* and *elderly*. The ruleset that specifies the behavior contains two rules, one phase change rule that specifies the transition from *child* to *adult*, and from *adult* to *elderly*. This means that whenever a person is in the *child* phase, they may only transition to the *adult* phase in the next world, or remain a *child*. They cannot transition to *elderly*.

## 5.4 Formal Semantics

Here, we discuss the semantics of OBSL by describing the language constructs in terms of a formal logic.

### 5.4.1 Structure

OBSL utilizes the same conceptualization of *world* and *history* as OntoUML's Alloy-based model simulations (see Section 4.3). A history is formally defined as:

- A set of worlds *W*;
- A relation *Succeeds*, defining the succession between the worlds in W. This relation is structured such that *W* is a branching-time Kripke structure, as used in the Alloy specification of OntoUML [7];
- A set of individual entities *E*.
- A set of universals *U*.

Predicates are defined to relate the individual entities to universals and worlds:

A predicate *Obtains*($w \in W$, $e \in E$) which is satisfied if and only if the individual *e* exists in world *w*.

For each universal, a predicate *InstanceOf*($w \in W$, $e \in E$, $u \in U$) satisfied if and only if the individual *e* is an instance of that universal in world *W*.

A predicate for each association (mediation, characterization, material association), *AssociationObtains*($w \in W$, $e_1 \in E$, $e_2 \in E$, $u \in U$ ), which is satisfied if and only if the association *u* holds between $e_1$ and $e_2$ in world W.

### 5.4.2 Rulesets

A ruleset is formalized as a predicate without parameters. A ruleset is satisfied if for each pair of consecutive worlds and each individual in the history, one of the transition rules is satisfied:

$$
\begin{aligned}
&\textit{Rulesetx}() \Leftrightarrow \\
&\quad \forall w, w' \in W, w' \text{ succeeds } w, \forall e \in E : \\
&\quad [\textit{Conditions}] \rightarrow \\
&\qquad \text{TransitionRuleX}_1 (w, w', e) \vee \ldots \vee \text{TransitionRuleX}_n (w, w', e)
\end{aligned}
$$

Equation 1: Semantics for rulesets

The *conditions* under which a ruleset applies depend on the category of the entity being modeled. These conditions are formalized as an open sentential formula (i.e., a partial expression). The intuition is that a ruleset only applies when the entity *changes.* Additionally, for dependent entities (phases, modes, qualities, relators), the behavior governing the bearing entity takes precedence.

For kinds and subkinds, the conditions are formalized as:

$$\text{Obtains}(w,e) \text{ xor } \text{Obtains}(w',e)$$

Equation 2: Conditions for kinds and subkinds

For phase partitions, the transition rules should only apply when the individual bearing the phases exists in both worlds, but not in the same phase. The first part of this condition means that transition rules for the behavior of the bearer of phases take precedence over transition rules for phases itself. In particular, it is possible to describe the deletion of the bearer of a phase in the ruleset for that bearer, without also having to specify this possible transition (from a phase to no phase at all) in the ruleset for the phase partition. Limiting the deletion of entities to only certain phases is done by including that phase in the description of the circumstances of deletion of the bearer.
The second part of the conditions for phase partitions (phases must be in different states) means that it is always possible for an individual to remain in the same phase. The transition rules only govern *changes*. Formalized:

With *bearer* the universal that bears the phase partition, and *partition*, the set of phase universals that compreise the partition:

InstanceOf (w, e, *bearer*)
∧ InstanceOf(w',e, *bearer*)
∧ (∄*p* ∈ *Partition*| InstanceOf (*w*, *e*, *p*) ∧ InstanceOf (*w'*,*e*, *p*) )

**Equation 3: Conditions for phases**

For intrinsic moments (modes and qualities), the transition rules only apply if the individual bearing that moment exists in both worlds. Again, this is so that the creation/deletion of the bearer is not limited by the behavioral rules of its intrinsic moments.

With *b* the individual that bears the mode or quality:

∃ *b* ∈ E |
(Obtains(w, *b*) ∧ Obtains(w', *b*))
∧ (Obtains(w, *e*) xor Obtains(*w'*,*e*) )
∧ (Characterizes(w, *e, b*) ∨ Characterizes(*w'*, *e, b*))

**Equation 4: Conditions for modes and qualities**

Relators are existentially dependent on their relata. Therefore, it should be possible for a relator to be deleted whenever one of its relata is deleted. To achieve this, the behavioral rules for relators only apply when none of its relata are deleted:

∄ $r_1$ ∈ E | (*Mediates*(w, *e, r*$_1$) ∧ ¬Obtains(w, $r_n$)
∧ …
∧ ∄ $r_n$ ∈ E | (*Mediates*(w, *e, r*$_1$) ∧ ¬Obtains(w, $r_n$)
∧ (Obtains(w, *e*) xor Obtains(*w'*,*e*) )

**Equation 5: Conditions for relators**

### 5.4.3 Transition rules

A transition is represented by a predicate, parameterized by a pair of worlds w, w' and an individual e, which represents the domain entity governed by the transition:

$$TransitionRule_x(w, w' \in W, e \in E)$$

Equation 6: Predicate for transition rules

The transition rule defines the entities shared between both situations, existentially quantified. The rule is satisfied if there exist entities such that the before-situation (appropriate parameters) obtains in *w*, the after-situation obtains in *w*'.

### 5.4.4 Situations

Situations are also defined as predicates, parameterized by a world W and individual entities.

$$Situation_x(w \in W, p_1 \in E, \ldots, p_n \in E)$$

Equation 7: Predicate for situations

In addition to the parameterized individuals, a situation may contain additional individuals, which only occur locally. These are existentially quantified.

$$\exists\ a_1 \in E, \ldots, a_m \in E :$$

Equation 8: Quantification for local variables

The individuals bound in a situation are disjunct:

$$disjunct(p_1, \ldots, p_n, a_1, \ldots, a_m)$$

Equation 9: Disjunct variables

A situation predicate is defined as a conjunction of the contents of the situation:

- Classification of an individual: $InstanceOfUniversal(w, e_i)$
- A relation: $AssociationObtains(w, e_i, e_j)$
- A subsituation: $Situation_y(w, \ldots)$ with the appropriate individuals passed as parameters.

Some elements of a situation description may be defined in a negative form:

- An individual stated to be absent: $\neg Obtains(w, e_i)$
- An individual not classified by a universal : $\neg InstanceOfUniversal(w, e_i)$
- A subsituation that should not obtain: $\neg Situation_y(w, e \ldots)$

### 5.4.5 Example

Consider a domain ontology containing the kind *person* and the phases *child*, *adult* and *elderly*. For these entities, we wish to define the behavior: "a person starts as a child". This is formalized as:

RulesetForPerson: $\forall w,w' \in W$, w' succeeds w, $\forall e \in E$ :
Exists(w, e) xor Exists(w',e) →
PersonCreationRule (w, w', e)
∨ PersonDeletionRule (w, w', e)

PersonCreationRule (w, w∈ W', e∈ E) : SNotPerson(w, e) ∧ SChild(w',e)
PersonDeletionRule (w, w'∈ W, e∈ E) : SPerson(w, e) ∧ SNotPerson(w',e)
SPerson(w, e) : InstanceOfPerson( w,e )
SNotPerson(w, e) : not(InstanceOfPerson( w,e ))
SChild(w, e) : InstanceOfPerson( w,e ) ∧ InstanceOfChild( w,e )

The behaviors "children can become adults" and "adults can become elderly" are behaviors that apply to the *age* phase partition. They are formalized as:

RulesetForAge: $\forall w,w' \in W$, w' succeeds w, $\forall e \in E$ :
InstanceOfPerson(w, e) and InstanceOfPerson (w',e)
and not (
(InstanceOfChild(w, e) and InstanceOfChild (w',e))
or(InstanceOfAdult(w, e) and InstanceOfAdult(w',e))
or(InstanceOfElderly(w, e) and InstanceOfElderly(w',e))
) →
AgeChangeRule1 (w, w', e)
∨ AgeChangeRule2 (w, w', e)

AgeChangeRule1 (w, w∈ W', e∈ E) : SChild(w, e) ∧ SAdult(w',e)
AgeChangeRule2 (w, w∈ W', e∈ E) : SAdult(w, e) ∧ SElderly(w',e)

SChild(w, e) : InstanceOfChild( w,e )
SAdult(w, e) : InstanceOfAdult( w,e )
SElderly(w, e) : InstanceOfElderly( w,e )

## 5.5 Discussion

### 5.5.1 Scope limitations

As OBSL is a proof-of-concept language, only a subset of OntoUML's categories is supported, namely kinds, subkinds, roles, phases, modes, qualities, relators, mediation relations, characterization relations, and material relations. Collections, part-whole relations and mixins are not considered.

OBSL's account of qualities is limited to values in the integer domain, since only these can be simulated with the Alloy analyzer. Furthermore, only single-value qualities are supported.

Formal relations are not represented either, except in a limited fashion, namely the direct comparison of integer values of a quality individual in separate situations. This is the bare minimum needed to be able to represent *some* behavior for qualities.

### 5.5.2 Clarity

Many existing languages for specifying behavior use a state-and-transition model. This is because it is relatively easy to formulate behaviors in terms of states and transitions, and conversely, it is not difficult to infer the behavior described by a state-transition model. State-transition rules in OBSL are relatively straightforward. The state-transition language is capable of describing most common behaviors as they occur in domains (see Chapter 7).

OBSL's situations and transitions can be clearly expressed in a graphical notation, which further aids the clarity of the language.

The organization of a behavior specification, with one rule-set for the behavior associated with each universal, guides the modeler towards the creation of a complete specification.

### 5.5.3 Expressivity

In several key places, OBSL has limited expressivity. This is an intentional choice in its design, to keep it relatively simple yet useful.

Behavior as described by OBSL is limited to changes occurring between consecutive worlds, and there is no mechanism for describing more involved behaviors such as complex multi-step processes or workflows. Similarly, OBSL has a limited capacity for expressing the *conditions* under which behavior can occur, again restricting itself to the pair of worlds in which the described behavior occurs.

OBSL's notion of composition is also quite limited, both at the level of situations and of the behavior specification as a whole.

Situations are chiefly composed through conjunction of individuals, with negation being used in some limited cases. Notably absent is a representation for *disjunction*, which could be used to describe alternative conditions under which a behavior may occur. Other logical operators (e.g., implication) are also omitted. While the inclusion of these operators would not increase the expressivity of situation descriptions (conjunction and negation can be used to model these other operators), a situation description that specifies complex conditions can be rather verbose without a richer composition structure.

The rules of an OBSL specification are also composed in a straightforward and limited fashion. Each domain element is governed by a rule set, which describes individual rules for the possible behaviors of that element. These rules are composed by *disjunction*: each occurrence of behavior for that element in the model is governed by (at least) one of the transition rules. There is no way to further group these rules, or to define an order of precedence among them.

The rule sets that govern each domain element are composed by conjunction: each rule set applies, in addition to the constraints derived from the OntoUML domain model. Again, no mechanism is provided to modify composition.

There is *some* precedence of rules defined in OBSL, and these are derived from the categories of the elements being modeled. Specifically, the rules for dependent entities, such as modes and phases, only apply in the context of their bearer. Even though there may be no rule that explicitly describes the deletion of a mode or phase, individual moments or phases may nevertheless be deleted when their bearer is deleted, if the deletion of the bearer is admissible according to the specification.

There are several reasons OBSL has such a limited capacity for defining composition. First, by limiting the expressivity in this manner, the language remains simple and therefore easy to understand and use. Second, a limited set of operators (conjunction and negation) is easily expressed visually, where a richer notion of composition would require more a complex representation. This issue is discussed in Chapter 6, which introduces the visual notation we developed for OBSL.

Finally, OBSL is a prototype. If more complex notions of composition prove to be necessary to define more realistic behavior specifications, these can be developed in later iterations.

### 5.5.4 Compatibility with OntoUML

OBSL's semantics is compatible with that of OntoUML as represented in Alloy. OBSL builds on the world structure for models as defined for OntoUML, and OBSL specifications rely on the constraints specified in OntoUML domain models to guide behavior. However, the state-transition paradigm for behavior used by OBSL is not a perfect fit for the semantics of OntoUML as implemented in Alloy, and this can be observed in several places.

The main issue is that an OntoUML model without an attending OBSL specification permits any kind of behavior allowed by OntoUML, whereas that same model with an *empty* OBSL specification (i.e. transition sets without transitions) permits virtually no behavior. Only by adding transition rules to the OBSL specification can a modeler allow more behavior. This runs contrary to the view of modeling used by OntoUML, where rules constrain the space of admissible histories (see Section 4.4).

For example, adding a relation between two universals in OntoUML constrains the admissible histories, by forcing that instances of these universals obey the cardinality constraints imposed by this relation. Conversely, adding a rule to an OBSL diagram *permits* a certain type of behavior that was not permitted before (such as the creation of some individual).

To alleviate this issue, an OBSL specification is generated from an OntoUML model with some transition rules already present, specifically so that the behavior of the newly generated specification is

closer to that of the original OntoUML model. There are some exceptions however (phases, qualities), and the modeler should be aware of this.

Because OBSL defines behavior as occurring between a pair of consecutive worlds, a behavior specification does not constrain the state of the initial world of a history. This may cause the initial world to contain a situation that is not reachable through applying the behavior specification on an empty first world. It is difficult to alter OBSL's or OntoUML's semantics such that this is prevented. A possible solution could be to constrain histories such that the initial world must be empty. However, the consequence is that generated histories will increase in size. The 'initial state' required to simulate a particular behavior can currently be achieved in a single world. If the first world of a history is required to be empty, it may take several consecutive worlds to arrive at the same initial state. This increase in complexity of the simulation may negatively impact performance.

### 5.5.5 Ontological concerns

OBSL's transition rules are not intended to have any ontological status themselves. They merely constrain the behavior of the entities they describe. For example, consider the kind *person* with phases *alive* and *deceased*. An OBSL phase change rule from alive to dead should not be interpreted as the specification of a *death* event. Rather, because alive and dead are phases, they are derived from intrinsic qualities of *person*. If these qualities are to be specified directly, this should be done in OntoUML. The function of OBSL is to say, in absence of an explicit definition of these qualities, they are constrained by OBSL such that the resultant behavior from those qualities is compatible with the behavior described by the OBSL.

It is possible to define behavior in OBSL that is incompatible with the dynamic aspects of categories as defined by OntoUML. For example, a specification without phase change rules for a phase partition results in the phases of that partition being effectively *rigid*, which contradicts OntoUML's definition of phases. Fortunately, using explicit visualization of anti-rigidity for OntoUML, it is possible to detect these incompatibilities.

OBSL can be used to simulate complex behaviors (see Chapter 7). By combining the structural rules of OntoUML (a disjoint and complete partition of roles) and the creation-deletion rules of OBSL, the case study models a 'phase-like' behavior for a group of roles. The resultant OBSL specification is not particularly clear, in the sense that noticing the ordering of these roles requires the modeler to examine three different rule sets. A solution to this problem is to adapt OntoUML to include a more elaborate account of relators, or even incorporate a philosophically grounded account of perdurants. OBSL could then be adapted to represent these constructs.

# 6 Graphical notation

One of the key characteristics of our approach is clarity of the behavior specification, both while designing it as while reading it. A suitable visual notation is a great aid in this process. This chapter describes the visual notation developed for OBSL.

Section 6.1 presents a theory for the creation of effective visual notations, which was used as a guide for developing OBSL. Section 6.2 presents the basic components of OBSL's visual notation, which are the symbols used to represent situations. Section 6.3 presents the structures aggregating these symbols into situations, behavioral rules and the complete behavior specification. Section 6.4 presents the visual notation for different types of negation. Section 6.5 provides a discussion of the properties of the resulting visual notation.

## 6.1 Background

In [8] the author defines a theory for the design of effective visual notations. This work is extensively sourced, and is based on research in a wide variety of fields, including communication, semiotics, graphic design, visual perception, and cognitive psychology. For a detailed discussion of this theory, we refer to the paper. The conclusions and recommendations proposed by this theory are helpfully codified in a set of nine design principles. The following sections summarize these principles.

### 6.1.1 Semiotic Clarity

An effective visual notation requires a 1-to-1 correspondence between elements of the visualization and the semantic constructs they represent. This principle is based on the same theoretical framework as OntoUML's (see section 2.3.1), but applied to visual notations instead of ontologies. The concepts are equivalent: mismatches between visualization and underlying semantics are classified as symbol *excess*, *overload*, *redundancy* or *deficit*, and the resulting issues are similar to those for ontologies.

The exception is formed by symbol deficit, since in visual notations it is often desirable to limit the complexity of diagrams. Not all language constructs should be represented in a visualization, especially when dealing with languages that have a lot of constructs.

### 6.1.2 Perceptual Discriminability

The symbols of a visual notation should be clearly distinguishable. The ability to discriminate between symbols depends primarily on the *visual distance* between them. Visual distance is defined as the number of *visual variables* in which two symbols differ. Visual variables are the building blocks of any graphical notation, summarized in Figure 14.

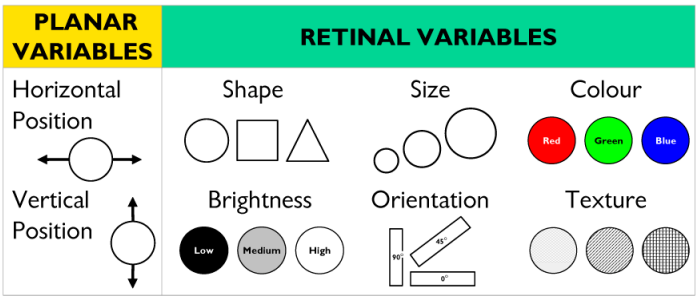



Figure 14: Visual variables [8]

A few factors further influence discriminability. First, the shape of symbols is the prime variable used to distinguish symbols. Second, by separating symbols along multiple variables (called redundant coding), they become more easily distinguishable. Third, by giving each symbol a unique value for at least one visual variable, they can be more readily identified. Finally, text is not a visual variable, and text is not a suitable method to use distinguish different symbol types. However, text can be used to distinguish individual symbols of the same type.

### 6.1.3 Semantic Transparency

By using symbols whose appearance indicates their meaning, they are more easily interpreted correctly. This can be applied to icons as well as relationships. Again, shape is the primary variable used to identify symbols.

Icons can use resemblance (a bull's-eye for 'Target'), common properties (Venn diagrams to show overlap), or metaphor (a thrash can for 'Delete').

Relationships can be represented using arrows, but also containment, overlapping symbols, or spatial location. The restrictions imposed by the choice of representation should correspond to those of the semantics (e.g., containment cannot be used to represent multiple inheritance).

A symbol that is a suitable mnemonic for what it represents (its meaning) is called *semantically immediate*. A symbol that offers no indication of its meaning, and thus requires memorization, is called *semantically opaque.* Finally, a symbol that has a different meaning than its appearance suggests is called *semantically perverse*.

### 6.1.4 Complexity Management

It is difficult to interpret an entire specification displayed in a single diagram. By including mechanisms to deal with the complexity of diagrams, they become more readily understandable.

Two key mechanisms are *modularization* and the use of *hierarchies*. Modularization covers the decomposition of a visual notation into separate, smaller diagrams. The use of hierarchies covers the organization of these diagrams into a structure. By *summarizing* diagrams in a higher-level visual notation, complexity is managed and a system can be understood in a top-down manner.

The management of complexity cannot be accomplished only at the level of syntax, but it requires that the underlying semantics provide a suitable notion of (de)composition.

### 6.1.5 Cognitive Integration

If a system is represented by multiple diagrams (for the purpose of complexity management), the user is required to integrate these diagrams in order to understand the whole system. A good visualization provides the tools to perform this integration. This includes an *overview* diagram that summarizes the system as a whole, and *navigational tools* that link the diagrams.

### 6.1.6 Visual Expressiveness

A visual notation should use the full range of visual variables in its representation. This is similar to the principle of discriminability, except applied to the diagram as a whole, rather than to individual symbols.

### 6.1.7 Dual Coding

Use text to complement graphics. Annotations (similar to comments in code) can aid understanding, if they are easily distinguished from the representation itself. Hybrid notations (using both text and graphics) can be useful.

### 6.1.8 Graphic Economy

The number of different symbols should be manageable. Humans can only easily distinguish between a limited number of categories (based on a single visual variable), so that interpreting more complex visual notations can become problematic. There are three ways to deal with this issue:

1. Reduce the complexity of the semantics.
2. Introduce *symbol deficit*: do not distinguish between certain types of element.
3. Increase visual expressiveness. By using more visual variables to distinguish between symbols, the amount of categories that can be distinguished increases.

### 6.1.9 Cognitive Fit

The suitability of a visual notation depends on the task it is aimed at, the level of expertise of the user, and the medium used to display the notation. For this reason, it may be beneficial to develop multiple representations.

### 6.1.10 Application

These nine principles are applied to the creation of a graphical notation for OBSL. Several factors limit the degree to which these principles are adopted:

The implementation of OBSL is a prototype that focuses on core functionality. Support for modularization through multiple diagrams, and the use of different types of diagrams for different purposes is not considered.

The technology used to implement an editor for OBSL diagrams has some inherent limitations. A limited number of shapes and connectors are supported, and it is difficult to create an automatic layout for diagrams.

The creator of OBSL has no background in graphic design.

## 6.2 Domain Elements

As a starting point for the examination of OBSL's visual notation, we examine its basic components, namely the representation of the contents of a situation.

### 6.2.1 Atomic Symbols

A situation description contains two categories of elements: Individuals corresponding to the Universals defined in an OntoUML domain ontology, and relationships that link these elements.

The Individuals we consider are instances of kinds, subkinds, phases, roles, qualities, modes, and relators. They can be related by generalization, characterization, mediation, and material association.

The individual types can be classified according to their category, distinguishing between substances and moments, considering rigidity, and considering the difference between inherent and mutual moments. By aligning the visual distance of elements with this hierarchy, the visualization conforms to the intuition of the modeler.

| Category | Inherence | Rigidity | Mutuality | Complexity |
|---|---|---|---|---|
| Kind | Substance | Rigid | Intrinsic | |
| Subkind | | | | |
| Phase | | Antirigid | Intrinsic | |
| Role | | | Mutual | |
| Relator | Moment | Rigid | Mutual | |
| Mode | | Antirigid | Intrinsic | Simple |
| Quality | | | | Complex |

Table 2: OntoUML categories and their properties

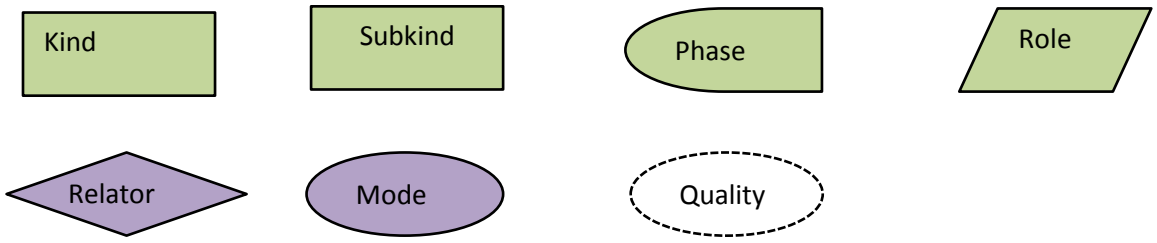



Figure 15: OBSL symbols for individuals

We aim at achieving semantic immediacy in that aspects of certain shapes are suggestive of their properties:

- Solid square shapes for Kinds;
- Phases are represented by a crescent shape, which is intended to relate to lunar phases;
- Color is used to distinguish substances from moments;

The palette of possible shapes is limited by the need for sufficient internal space to denote the name of the universal within the symbol, rather than as a label outside the symbol. This helps prevent visual clutter.

Furthermore, the symbols used represent the categories of OntoUML, and not the universals of the domain ontology being modeled. For example, in a model that includes a *car* and a *person* the same symbol is used to represent each individual, as they are both *kinds*. Using strongly figurative symbols for shapes introduces the risk of semantic perversity: a figurative symbol may be more reminiscent of a universal from the domain than the category it is meant to represent. Since this would reduce clarity, to avoid this risk OBSL aims at semantic transparency: it is better that the meaning of the symbols has to be memorized, rather than that they are erroneously interpreted as representing universals instead of a category of universals.

### 6.2.2 Instantiation and classifiers

An OBSL situation contains individuals, which are each classified by one or more universals. The universals classifying an individual must be related through generalization. When representing an individual, there are several alternative ways in which its classifiers can be represented. Figure 16 shows the alternatives.

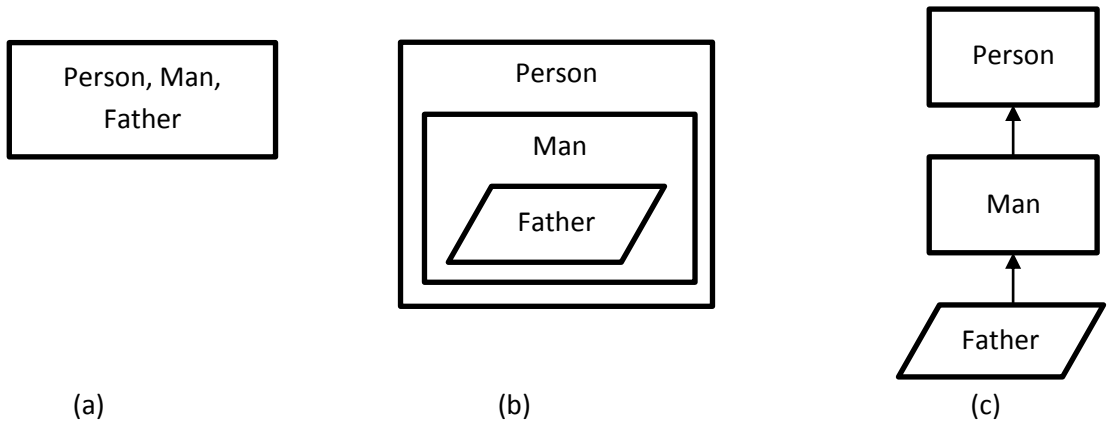



Figure 16: Representing individuals classified by multiple universals: a) textually; b) containment; c) relationships

Each of these alternatives has consequences with respect to the visualization. Alternative (a) is compact, but is purely textual, and cannot easily distinguish between category types. Alternative (b) requires all categories to be represented with a container that is roughly square-shaped, so that it can contain multiple categories efficiently, and is cumbersome to represent complex specialization hierarchies, since deeply layered boxes are visually confusing. Alternative (c), while clearly distinguishing classifiers, has a weaker representation of the *individual*: it is less clear that the three symbols represent properties of a single individual.

We opt for alternative (c).

The generalization relation should strongly connect its relata, visually speaking, as it represents an identity relation. To this end, we use a solid bold line. We include an arrow to indicate the direction of the generalization, similar to that of UML, and similar to that found in the OntoUML diagrams re-

lating the respective universals. As an additional constraint, we strive to keep the arrows that represent this relationship short, keeping the relata closely together in the diagram.

### 6.2.3 Qualities and characterization

The characterization relation between modes or qualities and their bearers, is represented by a solid arrow (Figure 17).

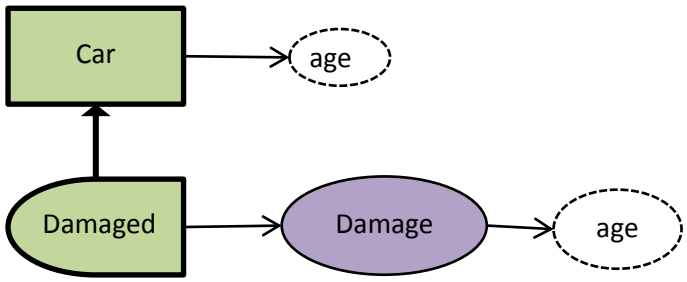



Figure 17: Representing qualities and modes

### 6.2.4 Mediation between relators and roles

A role represents an individual-as-a-participant in a certain type of material association. For example, the *father* role represents an individual that is a *father of* some child. Where the role represents the individual in the general sense, the relator represents the participating individuals with respect to that particular relator, since a relator represents the *qua individuals* of the participants.

Relators are the bridging element between different individuals in a situation. To represent the distance of the connection, we use a lighter line thickness, and a dotted line instead of the solid line used for identity and characterization.

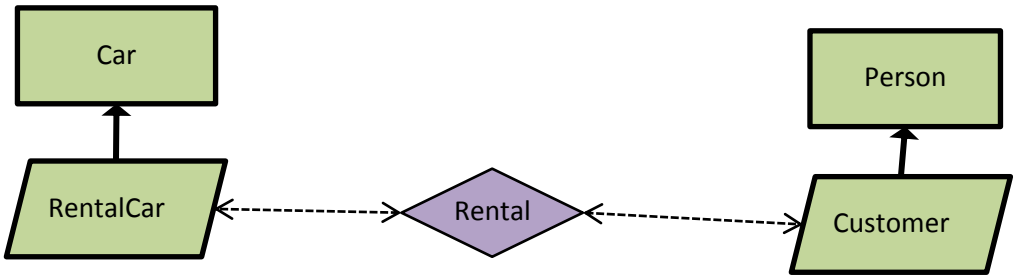



Figure 18: Mediation relations in OBSL

## 6.3 Situations, Rules, and Rule sets

Situations are a partial description of the state of a world. They consist of individuals and relations that together form a pattern, which should or should not be met by world states). In OBSL, Situations are represented by rounded rectangles. A situation is described by means of its domain elements, which are represented inside the symbol (rounded rectangle) that denotes the situation.

A transition rule consists of two such situations, a *before*- and *after*-situation, linked through a transition. The situations have a textual label. Because the direction of time is generally conceived of as

moving from left to right, the two situations that comprise a transition rule are ordered in this way, with the before-situation to the left, and the after-situation to the right.

The two situations of a transition rule are linked through elements that occur in both situations. Each transition rule has one *defining* individual, the individual that represents the type whose behavior is being defined. This entity is represented in both worlds, and linked through a broad dashed arrow. right. This link represents the flow of time from the before-situation to the after-situation.

Other elements that occur in both situations are linked through lighter, less broad dashed lines. To represent these elements as parameters of the situations they occur in, this parameterization is represented explicitly by a colored square.

The transition rule itself is displayed as a simple rectangle, with a background color that denotes the type of transition rule (green for creation, red for deletion, and blue for change).

A rule set is denoted by a grey rectangle, with the label denoting the Universal whose behavior is being described, and its category.

Figure 19 shows an example of a representation of a rule set.

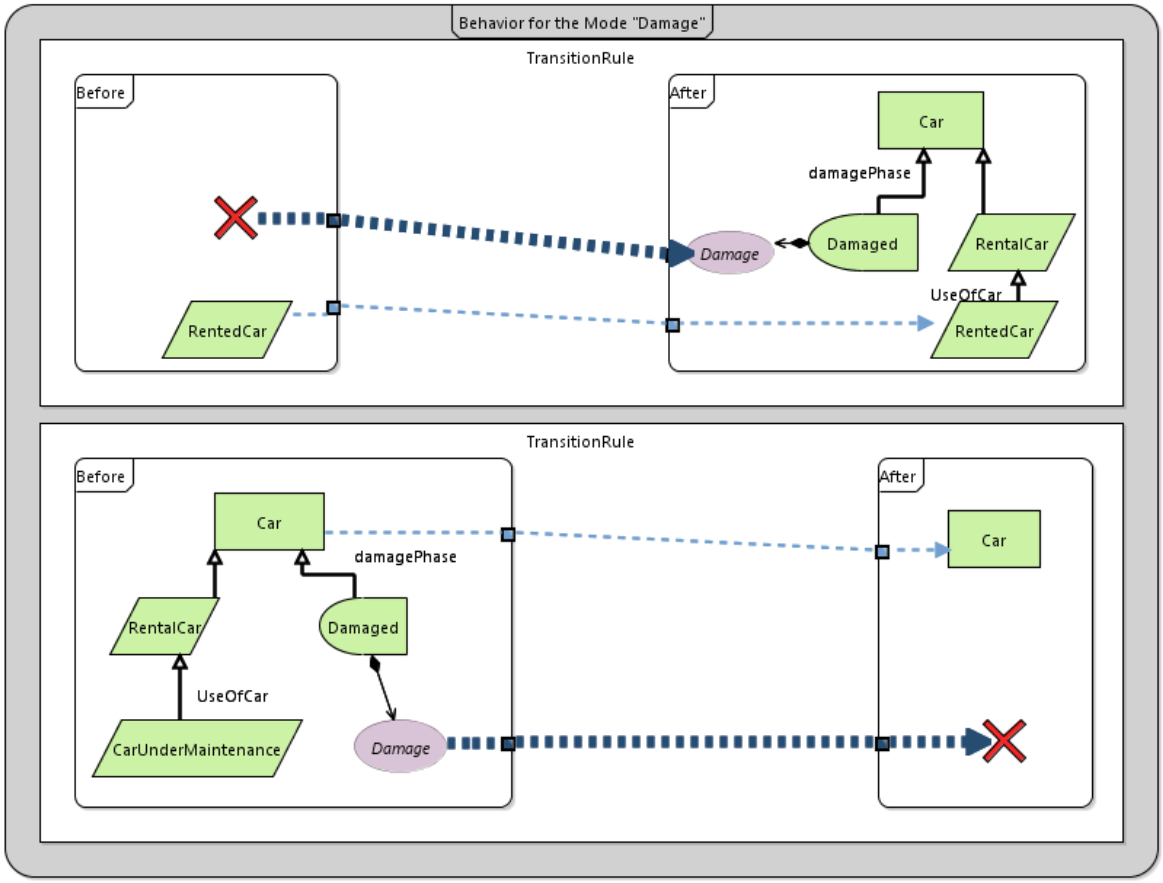



Figure 19: A transition rule set in OBSL

## 6.4 Representing absence or negation

There are multiple situations in which the notion of *negation* needs to be represented in OBSL. These are modeled as discussed below.

### 6.4.1 Absence of specialization

A substance individual may be classified by a universal, but not by a particular universal that is a specialization of that universal. The absence of the specializing individual is represented by using the red color to denote the specialization relation, the outline and the label of the specializing classifier. Additionally, the label is prefixed with 'not' (see Figure 20).

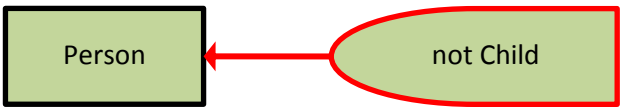



Figure 20: Absence of an aspect (phase, role, or subkind)

### 6.4.2 Future or past absence

An individual may be present in one of the two worlds, but not in the other. This is defined by representing the individual in one situation, and linking it to a symbol for absence that is shown in the preceding or following situation (see Figure 21).

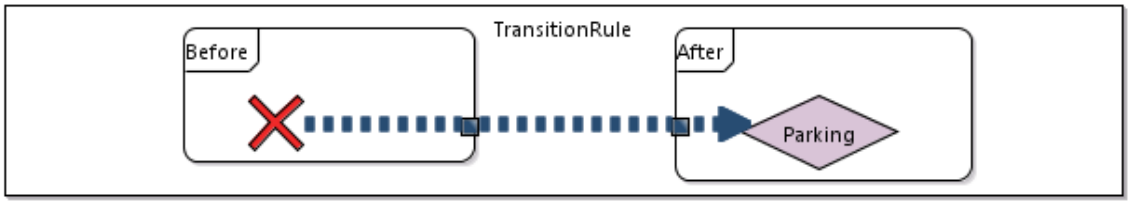



Figure 21: An individual that is present in one world, but not the other

### 6.4.3 Complex conditions

Sometimes we need to represent more complex conditions for behaviors that involve negation. To represent those conditions, OBSL uses sub-situations. The section of the condition that is negated is presented as a negated sub-situation, and the parent situation obtains if the sub-situation does not.

We represent the sub-situation as a red-bordered rectangle, labeled 'not'. Again, we use a dashed line with an explicit parameter marker on the situation border to represent a relation across situation borders.

Figure 22 provides an example of such a complex condition. It shows the visual representation of the situation where "there is a car that is under maintenance, and there is a garage, but it does not hold that this car is under maintenance in that particular garage."

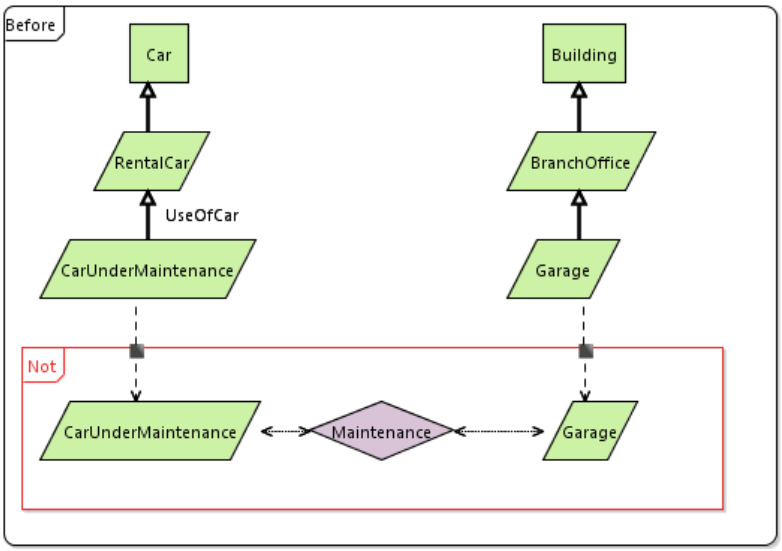



Figure 22: Example of negation through use of a sub-situation

This representation of negation is rather verbose, as classifiers in the sub-situation are linked to those in the parent situation through parameters, requiring the duplication of entities (in the example, CarUnderMaintenance and Garage). However, this verbosity is intentional. By forcing the modeler to explicitly and verbosely describe which classifiers obtain and which classifiers are part of the situation that must not obtain, the meaning of the specification is expect to be clearer.

By moving elements of the specification into the sub-situation, it is possible to make fewer claims about the nature of the individuals. For example, if we move all roles into the sub-situation, the situation obtains if “There is a car and there is a building, but it is not the case that the car is under maintenance in that building”

### 6.4.4 Negation of relations

Finally, we consider the representation of negation of relations, namely generalization, characterization, mediation, and material relations. Our analysis shows that including these options would not be beneficial for the reasons that we discuss in the sequel.

Classifiers that are represented separately in a single situation are considered to be disjoint (i.e., they do not classify the same individual), unless explicitly related through (a transitive closure of) generalization relations. Therefore, a classifier is not a generalization of another classifier unless this generalization is explicitly shown. There is no need to explicitly represent a negated *generalization*.

In the case of *characterization*, it is not possible to describe an intrinsic moment unless its bearer and the characterization relation are also represented. Since intrinsic moments *inhere in* a single bearer by definition, it follows that they do not inhere in any other individual than their bearer, and representing the negation of the characterization relation is therefore not meaningful.

The precise meaning of specifying a negated *mediation relation* is not immediately obvious. In order to represent a mediation relation, its relata must be represented as well. Therefore, using negation of a mediation relation, we can only specify situations in which a mediation relation is absent but its

*relata* are not. For example, consider Figure 23, which shows a situation in which there are a *person*, a *car*, and a *rental* seemingly relating these individuals, if not for the negation of one mediation relation (shown in red).

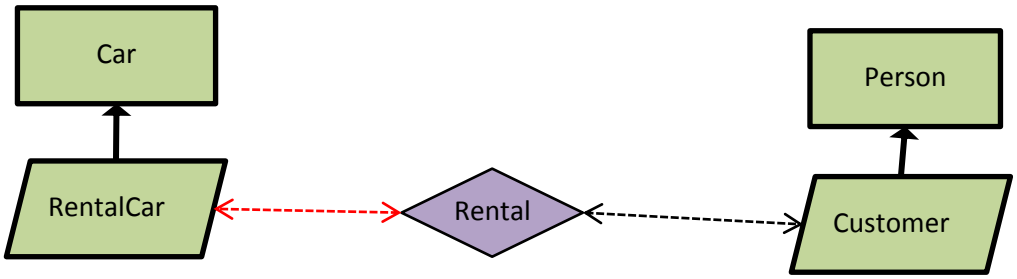



Figure 23: Possible representation of the negation of a mediation relation.

The situation shown in Figure 23 would obtain under the following conditions: there is a person renting some car. This is not the car shown in this diagram, but rather some *other* car, whose existence is implied by the existence of the Rental relator. The car shown in the situation should exist, and should also be rented by some *other* person, as the existence of the RentedCar role implies the existence of another Rental relator.

As shown, the semantic implications of using negation in this fashion are rather unintuitive, as they result in the implicit reference to additional entities (the other person and car) in worlds in which this situation obtains.

Moreover, it is not immediately obvious what the negated mediation relation is intended to express. We discern some alternative situations surrounding the mediation relation in which negation plays a part. Using the entities from Figure 24 as an example, we propose suitable representations of these situations. “A car exists, but is not rented” can be represented by negation of the RentalCar role. Variations of “A car exists, but is not rented by this particular person” can be represented by the use of sub-situations. As explained in Section 6.4.3, the use of sub-situations yields a more explicit representation of the semantics, making it more likely that the intended interpretation is conveyed.

To summarize, representing negation of mediation relations would be a case of construct excess: similar expressivity can be achieved through the use of other construct, whose meaning is clearer.

*Material relations*, finally, could also potentially be negated. However, they suffer from some of the same problems as representing the negation of mediation relations. Figure 24 shows a material relation, derived from the Rental relator shown in Figure 23.

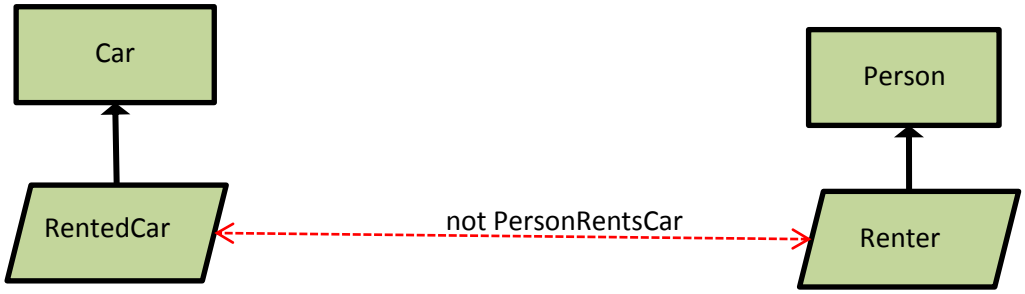


Figure 24: Possible representation of the negation of a material relation.

The issue with this representation is that material relations are defined between *roles*, and not the substances bearing them. Although the material relation is marked as negated, the roles are *not*. The result is similar to that of the previous figure: the situation of Figure 24 obtains only when the car is not rented by the person, but makes the additional requirements on the existence of another person and car related to those represented in the figure.

Ideally, when specifying the *absence* of a material relation, the situation should be agnostic towards the existence of other relations. Because agnosticism in OBSL is represented by the absence of an element, this would require representing the material relation without representing the roles. This conflicts with OntoUML's definition of material relations.

Again, sub-situations should be used to represent this behavior.

## 6.5 Discussion

The visual notation for OBSL was designed according to a theory for effective visual notations. On the whole, this has resulted in a notation that is clear enough to translate a conceptualization of behaviors into an OBSL specification (and vice versa).

There are a few weak points tough in OBSL's visualization, which we discuss below.

The use of abstract geometrical figures as shapes for OntoUML's categories is not ideal. The theory for visual notation specifically objects to these types of shapes, as they offer no mnemonic aids to allow a user to remember them more easily. However, as explained in Section 6.2.1, more figurative symbols run the risk of clashing with the concepts of a domain being modeled.

Another issue is the difficulty of establishing *identity* when reading a diagram. Individuals are represented as 'decomposed' into their classifiers. As a result, separate symbols in a single situation can depict aspects of a single individual (e.g., kinds and phases). The visual notation denotes this shared identity by connecting these separate symbols with an arrow, but it is still a potential source of confusion.

Similarly, two symbols in separate situations may represent the same individual as well, but in separate (consecutive) worlds. Again, the arrow-notation indicates shared identity, but similar confusion may occur.

The visual notation does not distinguish strongly enough between the defining elements and the constraining elements of a transition rule.

Summarizing, the visual representation is sufficient, although there are some areas in which improvements may be possible. Some of these limitations are caused by trade-offs inherent in the language being represented, while others are caused by limitations of the tool used for implementation, or the graphical design skills of the designer.

# 7 Implementation

This chapter describes the tools that we have implemented to support OBSL. This implementation can be found at http://code.google.com/p/bsl-for-ontouml/.

Section 7.1 provides an overview of the Eclipse Modeling Framework and related technologies, which were used to implement these tools. Section 7.2 discusses OntoUML's infrastructure, insofar as OBSL relies on it. Section 7.3 gives an overview of the architecture of our implementation. Section 7.4 discusses the OBSL metamodel and the transformation from OntoUML to OBSL. Section 7.5 discusses the visual model editor. Section 7.6 discusses the transformation process from the OBSL model to a partial Alloy specification. Section 7.7 provides a discussion of the created work.

## 7.1 Eclipse Modeling Framework

OBSL tools rely on the Eclipse Modeling Framework [9] (EMF) and related technologies for its implementation. EMF is a framework to support Model-Driven Engineering (MDE) in Eclipse. Its foundation is the modeling language ECore, which can be used to create (meta-)models. It is supported by the automated generation of model editors and model implementations.

EMF includes a number of other technologies based on ECore, among which are languages for model-to-model transformation languages, constraint-based model validation, specification of graphical editors. OBSL tools rely on the following EMF-based technologies:

- ECore ;
- Object Constraint Language [20] (OCL);
- Eclipse M2M Operational QVT [21], an implementation of Query/View/Transformation - Operational Mappings, an Object Management Group (OMG) standard for model transformation languages;
- Sirius [22], a specification language for the creation of graphical model editors in Eclipse;
- Acceleo [23], an implementation of the OMG MOF Model to Text Language standard.

## 7.2 OntoUML Infrastructure

OBSL tools have been built upon existing tool support for OntoUML, namely the OntoUML infrastructure [24]. This infrastructure is based on the OntoUML reference metamodel, which is an ECore-based representation of the OntoUML metamodel. This reference model is based on the ECore-representation of UML 2.0, and extends it with OntoUML's constructs by including OCL-based syntactic constraints. This reference metamodel is used as the standard representation of OntoUML by its EMF-based tools.

Our OBSL tools create behavior specifications for OntoUML models based on this reference metamodel.

## 7.3 OBSL Tool Infrastructure

The OBSL tools support the creation of new behavioral specifications based on OntoUML models, the modification of these behavior specifications using a graphical editor, and the translation of a behavior specification to an Alloy specification. Figure 25 summarizes the process of creating a behavior specification with OBSL.

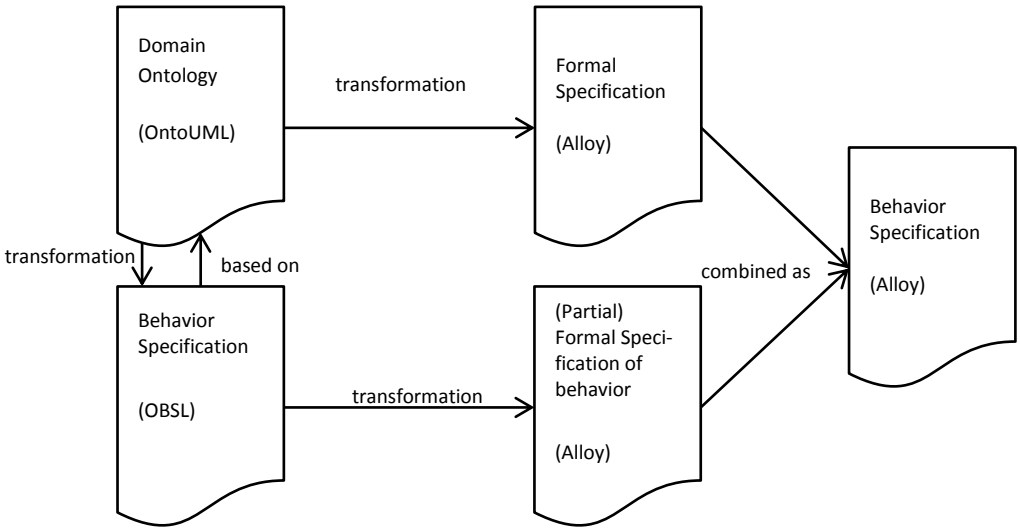



Figure 25: Artifacts in the behavior specification process

This process begins with a domain ontology created with OntoUML, and the specification of its semantics in Alloy (see Section 4.2). These can be created with other available OntoUML tools. An OBSL behavior specification is based on an OntoUML domain ontology, represented as an instance of the aforementioned OntoUML reference metamodel. A new, empty OBSL specification can be created based on an OntoUML model. OBSL models are defined according to a metamodel, specified in ECore. The transformation from an OntoUML model to a new behavior specification is specified in QVT. Section 7.4 describes the OBSL metamodel and transformation in further detail.

The OBSL graphical editor defines the visual representation for OBSL models, as well as the operations that can be used to modify the behavior specification. This graphical editor is implemented using Sirius. Section 7.5 details the editor.

Finally, the behavior specification can be transformed to an Alloy specification, which, combined with the Alloy specification for the OntoUML model, results in a behavior specification that can be simulated with Alloy. The transformation from an OBSL specification to Alloy is a multistage process in which the specification is first transformed to an intermediary representation as a model, and then to a textual Alloy specification. Section 7.6 describes the implementation of these transformations.

OBSL is implemented using principles of model driven engineering (MDE). We define a metamodel for OBSL, and transformations from OntoUML to OBSL, and from OBSL to Alloy. These transformations are linked in a transformation chain as shown in Figure 26.

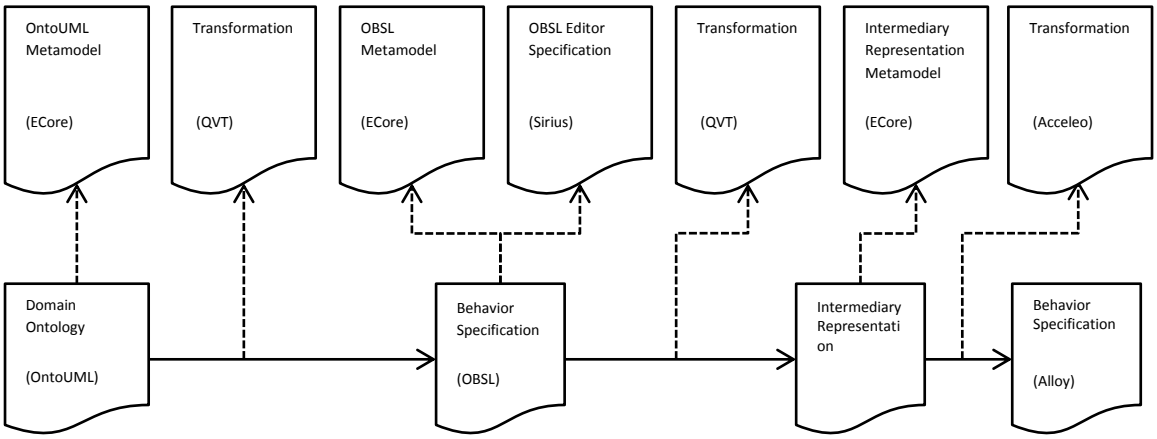



Figure 26: Transformation chain of the OBSL tooling

## 7.4 OBSL

The OBSL abstract syntax is specified as an ECore metamodel. This abstract syntax is a relatively straightforward description of the OBSL language as presented in Chapter 5.

One important issue with this metamodel is what is called the deep metamodelling problem. OBSL is based on an OntoUML domain ontology, and it must refer both to the universals of that ontology, by specifying behavioral rules for them, and to the instances of those universals, which are used to specify the behavior, in the form of situations. EMF does not support the explicit representation of instantiation in a model. Therefore this instantiation relation is represented by a plain *reference*, i.e., Individuals in an OBSL model *refer to* Universals in an OntoUML model.

The consequence is that the constraints described by an instantiation relation (instances *conform to* universals) are not automatically applied. Instead, OCL has to be used to define these constraints.

For example, consider an OntoUML model with the kind *person* and the phase *adult*, and a corresponding OBSL model with a situation in which there is an individual kind and phase. These individuals are meant to be *instances* of the OntoUML universals. An actual instantiation relation would constrain these instances so that an instance of the phase adult *must* refer to an instance of the person kind.

A new OBSL specification can be generated from an OntoUML model that conforms to the OntoUML reference metamodel. Such an 'empty' specification contains a rule set for each entity in the OntoUML model whose behavior should be specified (as described in Section 5.1), and a series of appropriate 'default' transition rules.

## 7.5 Editor

The OBSL graphical editor has been implemented using Sirius. This editor contains the visual notation for OBSL models, in accordance with the notation described in Chapter 6, and a series of commands that can be used to add elements to a specification.

The commands defined in the graphical editor are such that only valid models can be created. For example, it is only possible to create a specialization relation between classifiers in a situation if the corresponding OntoUML universals are related by specialization.

Unfortunately, the Sirius framework defines extensive default behavior for model editors, which is difficult to disable. Particularly, it is possible to delete elements of a specification, even if this is not intended and would result in an invalid model.

## 7.6 Transformation to Alloy

The OBSL metamodel has been designed to match the Sirius specification, which has some differences from the Alloy specification. The main difference is found in its description of individuals and classifiers.

Consider, as an example, an individual kind *person*, who is a *man* and is *alive*. In OBSL, this is represented by three individuals, each classified by their respective universal, and related by generalization. This is not the most ontologically accurate representation of individuals, but it is used because it matches the visual representation used by OBSL. Because we are interested in the behavior derived from each *aspect* of an individual, individuals are explicitly shown in this decomposed representation in OBSL. This makes it easier to discern which aspect is involved in which behavior.

Alloy, conversely, represents this group of entities as a single variable, and three statements of classification (called *binding*). In order to make the translation from OBSL to a representation in Alloy, appropriate variables must be created, and the elements of a situation must be linked to these variables.

Additionally, variables, universals, etc. must receive appropriate names, compatible with those used by the Alloy specification for OntoUML models.

The transformation process first transforms the OBSL model to an intermediary representation, which more closely resembles the Alloy specification. This transformation introduces and names variables, and links classifiers, variables and parameters. It also removes the explicit references to the OntoUML model, retaining only the names of OntoUML universals.

A model-to-text transformation then translates this intermediary specification to a textual specification in Alloy that closely resembles the formal semantics of OBSL as defined in Section 5.4.

An alternative model-to-text transformation is included, which adds the additional constraint that a history must include an explicit example of each specified behavior rule. This method is similar to the explicit visualization of anti-rigidity constraints for OntoUML models [7]. By using this option, a modeler can more easily examine the correctness of his behavior specification, since if no valid history can be generated, then the modeler can conclude that a conflict exists in one of the behavior rules.

## 7.7 Discussion

OBSL is supported by a set of tools, namely a graphical editor, capabilities to process existing OntoUML models, and the capabilities to translate an OBSL specification to Alloy.

The transformation to an Alloy-based formalization integrates with the available formalization of OntoUML models [7], which provides the option to simulate domain models with the defined behavior. This helps modelers create a behavior specification that conforms to their intentions.

Our OBSL tools implementation is a prototype. The focus is on the core functionality and the clarity of the language and its visualization. Several issues remain unaddressed:

- OBSL's specifications are based on OntoUML models, and maintain a link to the original OntoUML specification. OBSL specifications are not resilient to changes to the underlying OntoUML models.
- Robustness of the implementation has not been a focus of this prototype. The validation of model correctness through appropriate OCL constraints is an area for further work.
- OBSL's implementation generates an Alloy specification based on an OBSL specification. These specifications are intended to be used by the Alloy Analyzer to generate example models that conform to the behavior specification. The generation of example models can be computationally expensive. The performance characteristics of the Alloy specifications created with OBSL by our tools are not the focus of our work.

# 8 Case Study

This chapter illustrates the use of OBSL to define a behavior specification for a domain model. Section 8.1 describes the domain model. Section 8.2 demonstrate the specification of behavior for this domain model, by proposing intended behavior, showing the specification in OBSL. Section 8.3 shows the representation of the behavior specification in Alloy. Section 8.4 shows a simulation of the behavior specification. Section 8.5 discusses the issues encountered when defining this behavior specification.

## 8.1 Domain model

The domain model used in the case study models the renting of cars to people (see Figure 27). It contains four kinds:

- Persons, who can rent cars;
- Buildings, used for storage or maintenance of cars;
- Companies, who may own cars and buildings;
- Cars, which be damaged or undamaged (a complex quality represented as a mode), and may be used as a rental car by a company.

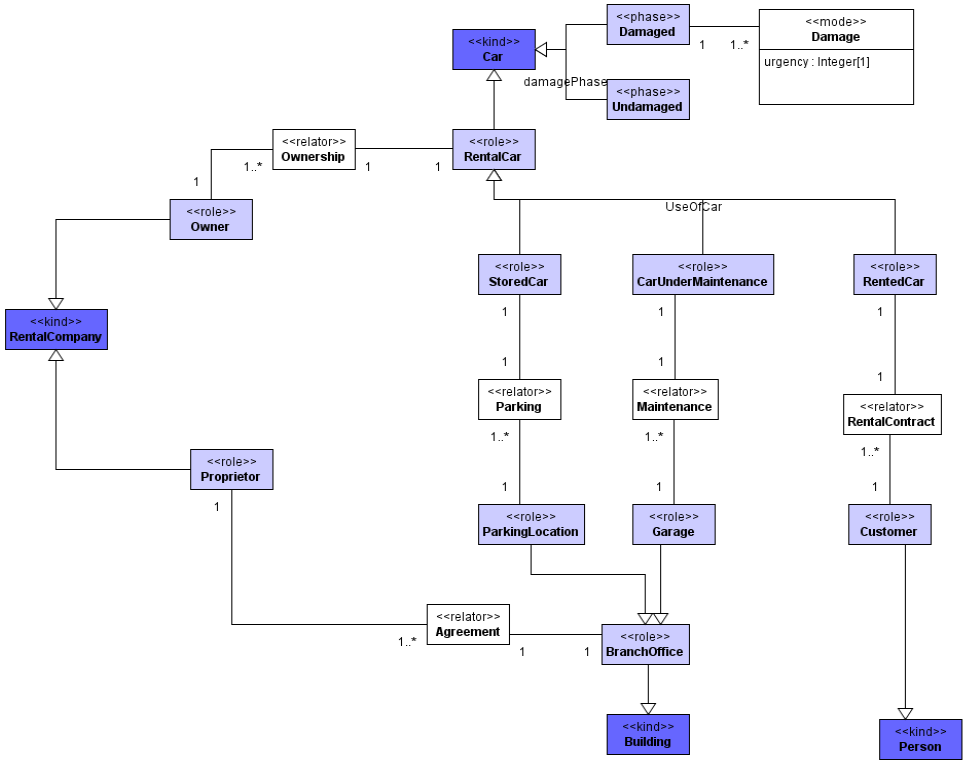



Figure 27: OntoUML model of a rental company

## 8.2 Behavior specification in OBSL

The following behavior is modeled for this domain ontology:

- Cars only accrue damage when they are rented;
- Cars can be repaired at a garage. They can only be sent to a garage when they are damaged, and can only leave when they have been repaired;
- Only undamaged cars may be rented.
- Persons are not created unless they are customers, and not deleted unless they are no longer customers.

In order to focus the simulation on these particular behaviors, we have chosen not to define the creation and deletion of cars, companies, buildings, etc. This means that when one of these entities exists in the history, they exist in every world of that history.

We describe the behavior specification for each universal in turn.

### 8.2.1 Persons

As an example of a simple set of behavior specification rules, we use the creation and deletion rules of the kind *person*. We specify the following behaviors:

- When persons are created, they must play the role of customer.
- Only persons that are no longer customers can be deleted.

The graphical notation for specifying these behaviors in OBSL is shown in Figure 28.

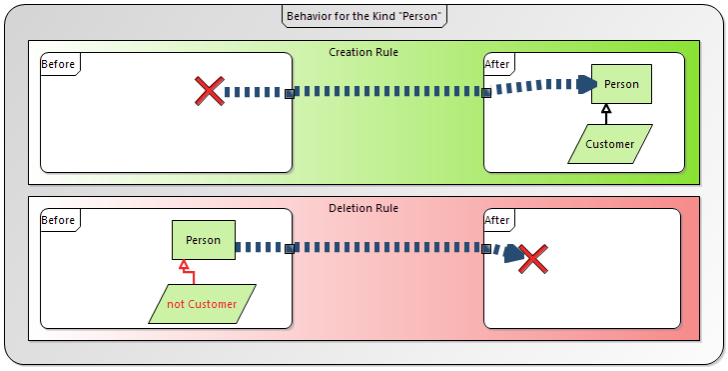



Figure 28: Behavior for persons

### 8.2.2 Non-changing individuals

Cars, buildings, and rental companies are defined as unchanging. The relators that relate a rental company to its cars and buildings are also defined this way. In OBSL, this means that no transition rules are specified in the rules sets that govern these universals (see Figure 29).

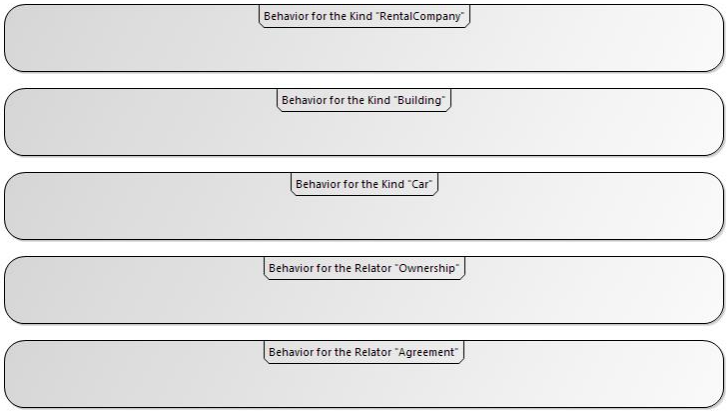



Figure 29: Rule sets for Kinds and some relators

### 8.2.3 Phase Partition

The damage phase partition of car has the following behavior associated with it: a car may change from damaged to undamaged, or vice versa. This behavior is specified in OBSL as shown in Figure 30.

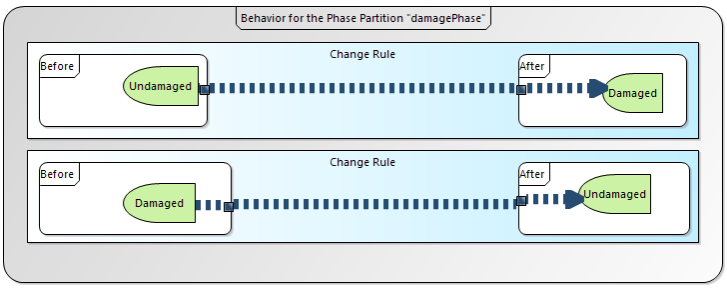



Figure 30: Behavior for a phase partition

### 8.2.4 Mode

The constraints on how damage can be accrued and removed are specified in the transition rules for the creation and deletion of the mode *damage*. The intended behavior is that damage can only occur when a car is being rented, and that damage can only be removed when a car is being maintained. This is specified in OBSL as shown in Figure 31.

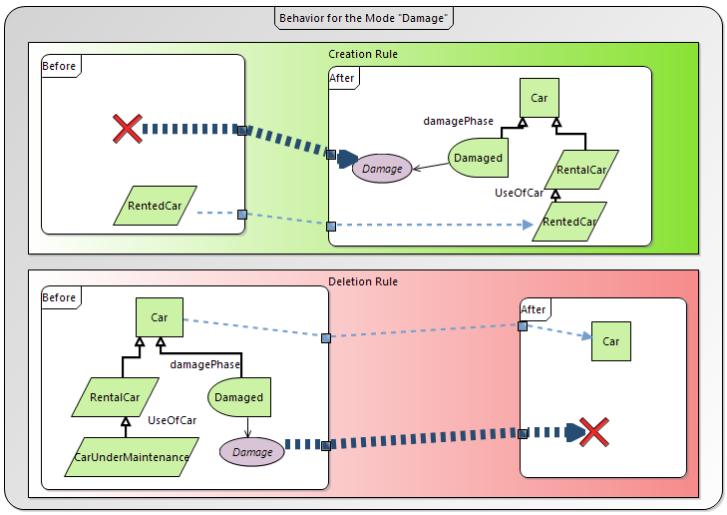



Figure 31: Behavior for the mode damage

### 8.2.5 Relators

The relators maintenance, Parking, and RentalContract together specify the more interesting behavior of the domain model.

The behavior for the RentalContract relator is specified as:

- Only undamaged cars may be rented to customers;
- The rental of a car may be terminated at any time.

These behaviors are expressed in OBSL as shown in Figure 32.

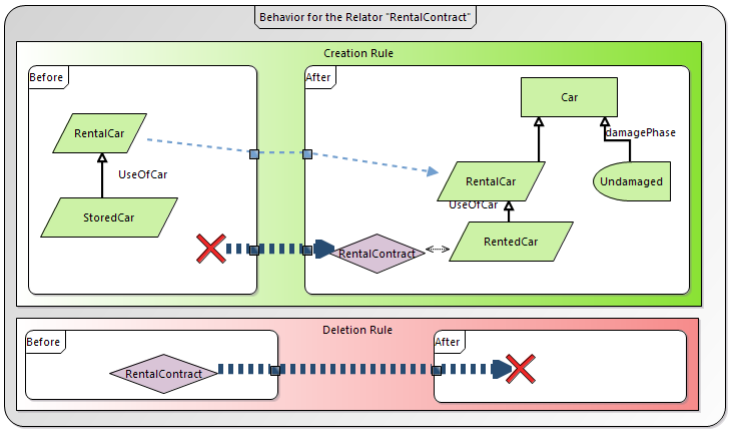



Figure 32: Behavior for the rental of a car

The behavior for the Maintenance relator is specified as:

- Only damaged cars may be sent for maintenance;
- Cars may only leave maintenance when undamaged.

These behaviors are expressed in OBSL as shown in Figure 33.

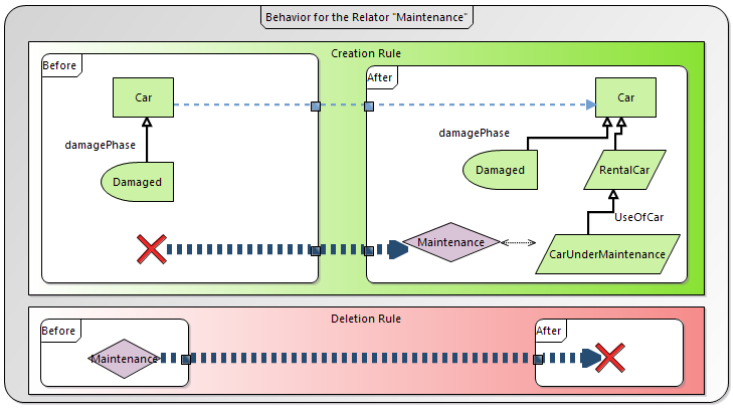



Figure 33: Behavior for the maintenance of cars

The behavior for the Parking relator is specified as:

- Cars may be parked at any time;
- Car may only leave storage when they are rented.

These behaviors are expressed in OBSL as shown in Figure 34.

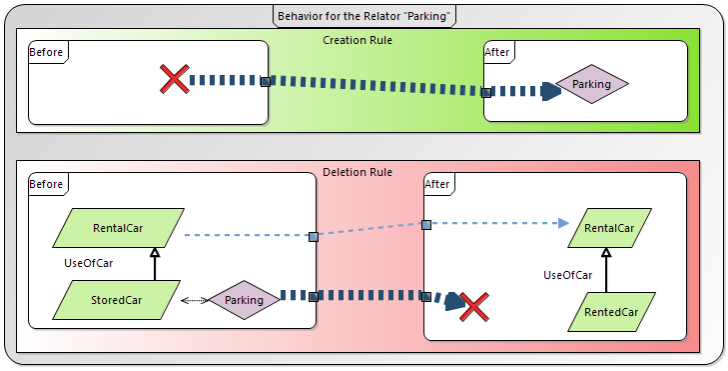



Figure 34: Behavior for the storage of cars

## 8.3 Representation in Alloy

The OBSL specification can be translated to a textual behavior specification in Alloy. A fragment of this specification is shown in Listing 1, which depicts the Alloy specification for the behavior of Persons, as specified in Figure 28.

```
fact PersonRuleset  {
    all w, w' :World, p2:Object | (
        (
            (w' in w.next)
            and ((p2 in w.Person) iff (p2 not in w'.Person))
        ) => (
            ((w.CreatePersonRule_before[p2] and w'.CreatePersonRule_after[p2]))
            or ((w.DeletePersonRule_before[p2] and w'.DeletePersonRule_after[p2]))
        )
    )
}
pred World.CreatePersonRule_before[p2:Object] {
    (p2 not in this.Person)
}
pred World.CreatePersonRule_after[p2:Object] {
    (p2 in this.Person)
    and (p2 in this.Customer)
}
pred World.DeletePersonRule_before[p2:Object] {
    (p2 in this.Person)
    and (p2 not in this.Customer)
}
pred World.DeletePersonRule_after[p2:Object] {
    (p2 not in this.Person)
}
```

Listing 1: Alloy specification for the behavior of persons

## 8.4 Simulation

We translated the OBSL behavior specification to a textual Alloy specification with our tools. We used the transformation that not only generates constraints that apply the behavior, but also requires histories to show an example of each behavioral rule that is specified.

We append this specification to the Alloy specification generated from the domain model. Using the Alloy Analyzer, we generate an example history from this specification. The result is shown in Figures 34-36.

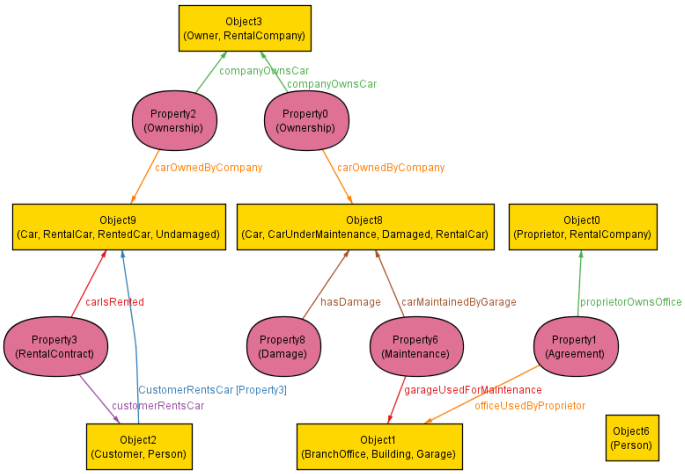



Figure 35: Alloy simulation: Past world

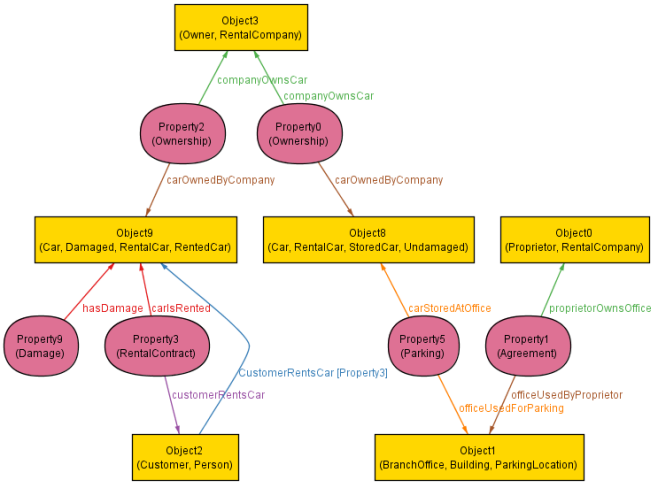



Figure 36: Alloy Simulation: Current World

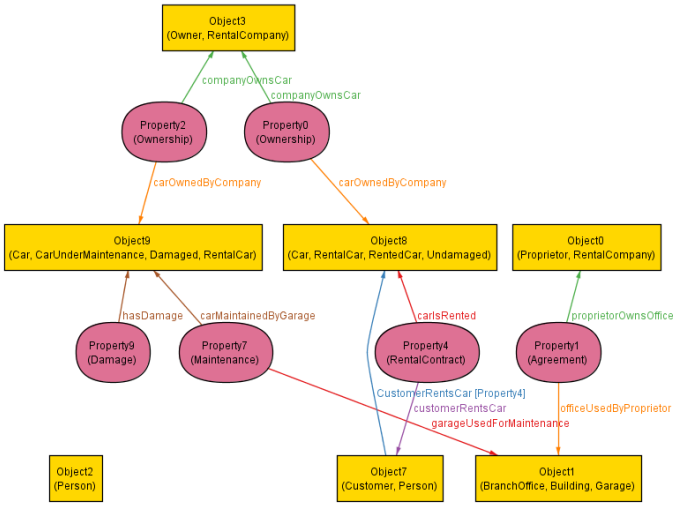



Figure 37: Alloy Simulation: Future World

## 8.5 Discussion

This case study demonstrates the capabilities of OBSL as a tool for behavior specification.

The translation from a concept of behavior to the graphical description of OBSL has proven to be fairly straightforward in each case.

The grouping of behavior according to rule sets guides the modeler in two ways. First, when trying to create a behavior specification for each type of entity in a domain, a modeler can simply specify each rule set in turn. When each rule set has behavior associated with it, the modeler knows he has considered each type of entity. Second, when a modeler conceives of a particular behavior he wishes to specify, placing it in a particular rule set makes it easier to retrieve that behavior among the other behaviors contained in a specification.

In some cases, related behaviors are spread across multiple rule sets, making it difficult to see their association. In the example specification shown in the case study, three relators (*maintenance*, *rentalcontract*, and *storage*) are defined as a *partition* in the OntoUML model, since a rental car plays one of these roles at any time. For this reason, the behaviors associated with these relators are related, i.e., when one relator is deleted, another must be created. This relation between behaviors is similar to what occurs with phase partitions. The behavior specification of phase partitions takes this into account through phase change rules. The rules that govern the behavior of these partitioned relators are not grouped in any way. Without referencing the domain model, a modeler cannot discern that these behaviors are related.

The generation of examples with the Alloy Analyzer is fast (seconds on a personal computer). Only in case a behavior specification is invalid and no examples can be generated the Alloy Analyzer takes longer to report this.

The generation of examples is useful, in that it can show possible configurations of the world that the modeler did not intend. The example model generated for the case study shows such an issue. In Figure 35, a car is under maintenance at a garage that is owned by a different rental company than the one that owns the car. This issue can be corrected by including the rental company in the behavior specification for the *maintenance* relator.

The generation of explicit examples of each transition rule is useful, in that it gives an overview of the possible behaviors displayed by the entities of a domain. It also serves to demonstrate that each behavior can occur.

Not all issues with a behavior specification are easy to detect. In the case study, the behavior related to the *damage* mode relies on the behavior related to the *damagePhase* phase partition. If we were to remove the behaviors associated with the phases from the specification, the behavior specification would still be valid, and an example of each behavior can still be generated. However, we have introduced a subtle problem: it is not possible to delete a damage mode if that is the last damage mode present on a car. In order to detect issues of this kind automatically, we would have to write Alloy rules that verify that whenever the *circumstances* for a transition rule occur, the behavior can *possibly* occur. To describe *possibility*, branching worlds can be used, but in that case, branches will

be created for each possible behavior of each entity at each point in time. The models generated from such a specification would be very large. It would not be feasible to use Alloy to evaluate such a specification.

# 9 Conclusions

This chapter discusses the academic contributions of the work presented in this thesis (Section 9.1), and suggests directions for improvements and further research (Section 9.2).

## 9.1 Contribution

In summary, this work provides the following contributions:

Chapter 4 analyzes OntoUML/Alloy as an ontology that describes dynamic aspects of a domain. We conclude that although OntoUML can express certain dynamic aspects through its *categories*, it lacks mechanisms to describe the behavior of domain entities that stems from the particular nature of those entities (i.e. from the domain).

Chapter 5 describes a behavior specification language that can describe the dynamic aspects of domain elements that are not expressible in OntoUML. By focusing on a *simple* language, we examine the feasibility of writing an adequate behavior specification without necessarily requiring knowledge of (temporal) formal logic.

Chapter 6 describes a visual notation for this language. By basing the design of this language on a theory of visual notation focused on cognitive effectiveness, we again focus on allowing the modeler to specify behavior intuitively, without necessarily having to think in terms of formal semantics.

Chapter 7 describes the prototype implementation in EMF. This tool allows modelers to create a specification based on an existing OntoUML model, and to convert this specification to a formal description in Alloy that is compatible with the Alloy-based formalization created for OntoUML. Though the prototype is integrated with OntoUML and Alloy tools, the OBSL and its tools are not only usable on existing models, but they also allow specified behaviors to be simulated and automatically verified for correctness and absence of conflicts.

Chapter 8 demonstrates the use of OBSL by applying it to an example case. This serves as an illustration of the language, as well as a demonstration of its usability.

## 9.2 Future Work

We identified three main areas in which future work can be performed by building upon the results of this thesis: improvements of the tools, integration with other methods for behavior specification, and the development of a more elaborate account of behavior in OntoUML.

There are many possible additions to the specification language presented in this thesis. Some of OntoUML's concepts were omitted. By adding support for mixins, part-whole relations, and (formal) associations, OBSL could be applied to any OntoUML model, rather than a small subset.

The tool itself is a prototype. The models used by OBSL can be made more robust by adding constraints. The graphical editor can be made more robust by defining a more complete set of tools in Sirius. This would include allowing drag-and-drop between different situations, preventing the deletion of entities that should not be deleted (e.g. rule sets), more accurately limiting the creation of associations only to valid associations, and defining validation rules that signal issues with the specification.

OBSL aims to describe behavior in simple terms, trading expressivity for ease of use. However, there are cases in which a modeler would need a more expressive language to describe behavior, e.g., when defining formal relations. Such a language for OntoUML, which is based on OCL enriched with temporal operators from CTL, is currently under development. It may be possible to integrate this language with OBSL, creating a hybrid approach in which OBSL is used to define the 'groundwork' of simple behaviors, and the more expressive language is used where necessary to define those behaviors that cannot be expressed in OBSL.

Finally, as discussed in Section 5.5.5, OntoUML's relators provide a limited structure for specifying behavior in a domain. By extending OntoUML's account of behavior to allow composition or partition in some way, OBSL could allow the representation of behavior for a group of conceptually related relators.

## References


[1] John Mylopoulos, "Conceptual Modeling and Telos," 1992.

[2] Olivé, *Conceptual Modeling of Information Systems*.: Springer-Verlag, 2007.

[3] R Weber, *Conceptual Modeling of Information Systems*., 1997.

[4] Giancarlo Guizzardi, *Ontological Foundations for Structural Conceptual Models*., 2005.

[5] Object Management Group. (2003) UML 2.0 Superstructure Specification. [Online]. http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/

[6] N. Guarino and C. Welty, "A Formal Ontology of Properties," in *Proceedings of the ECAI-2000 workshop on Applications of Ontologies and Problem-Solving Methods.*, 2000.

[7] A.B. Benevides, G Guizzardi, B.B. Braga, and J.A. Almeida, "Validating Modal Aspects of OntoUML Conceptual Models Using Automatically Generated Visual World Structures," *Journal of Universal Computer Science*, vol. 16, no. 20, pp. 2904-2933, 2010.

[8] D Moody, "The “Physics” of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering," *IEEE Transactions on Sofware Engineering*, vol. 35, no. 6, pp. 756-779, 2009.

[9] Eclipse Foundation. Eclipse. [Online]. http://www.eclipse.com

[10] Giancarlo Guizzardi, "On Ontology, ontologies, Conceptualizations, Modeling Languages, and (Meta)Models," in *Databases and Information Systems IV*, 2007, pp. 18-39.

[11] V. de Carvalho, JPA Almeida, and G. Guizzardi, "Using Reference Domain Ontologies to Define the Real-World Semantics of Domain-Specific Languages," in *Advanced Information Systems Engineering*, vol. 8484, Thessaloniki, 2014, pp. 488-502.

[12] J. M. ‘ Spivey, *Understanding Z*.: Cambridge University Press, 1988.

[13] I. Horrocks, P. Patel-Schneider, and F. van Harmelen, "From SHIQ and RDF to OWL: the making of a web ontology language," *Journal of Web Semantics*, vol. 1, no. 1, pp. 7-26, February 2003. [Online]. http://www.w3.org/TR/owl-ref/

[14] P. Chen, "The entity-relationship model: Towards a unified view of data," *ACM Transactions on Database Systems*, vol. 1, no. 1, 1976.

[15] Guizzardi, "Grounding Software Domain Ontologies in the Unified Foundational Ontology (UFO): The case of the ODE Software Process Ontology," in *IDEAS*, 2008.

[16] Armstrong, *Universals: An Opinionated Introduction*., 1989.

[17] D Jackson, "Alloy: a Lightweight Object Modelling Notation," *TOSEM (Transactions on Software Engineering and Methodology)*, vol. 2, no. 11, pp. 256-290, 2002.

[18] B. Henderson-Sellers, "Bridging metamodels and ontologies in software engineering," *Journal of Systems and Software*, vol. 2, no. 84, pp. 301-313, 2011.

[19] J Allen, "Maintaining knowledge about temporal intervals," *Communications of the ACM*, vol. 26, no. 11, pp. 832-843, November 1983.

[20] Object Management Group. Object Constraint Language. [Online]. www.omg.org/spec/OCL/2.2

[21] (2014, October) Eclipse QVTo. [Online]. http://www.eclipse.org/mmt/?project=qvto

[22] (2014, October) Sirius. [Online]. http://www.eclipse.org/sirius/

[23] (2014, October) Acceleo. [Online]. https://www.eclipse.org/acceleo/

[24] (2014, October) OntoUML Infrastructure. [Online]. https://code.google.com/p/rcarraretto/

[25] E.J. Lowe, *The Four-Category Ontology: A Metaphysical Foundation for Natural Science*. Oxford: Oxford University Press, 2006.

[26] Giancarlo Guizzardi, Gerd Wagner, Ricardo de Almeida Falbo, Renata S.S. Guizzardi, and João Paulo A. Almeida, "Towards Ontological Foundations for the Conceptual Modeling of Events," , 2013.

[27] S. Bechhofer et al. (2004, February) OWL Web Ontology Language Reference. [Online]. http://www.w3.org/TR/owl-ref/

[28] A.L. Opdahl and B. Henderson-Sellers, "Ontological Evaluation of the UM LUsing the Bunge–Wand–Weber Model," *Software and Systems Modeling*, vol. I, no. 1, pp. 43-67, 2002.

[29] T. P. Sales, P. P. F. Bardcelos, and G Guizzardi, "Identification of Semantic Anti-Patterns in Ontology-Driven Conceptual Modeling via Visual Simulation," in *4th International Workshop on Ontology-Driven Information Systems*, 2012.

[30] B Heller and H Herre, "Ontological Categories in GOL," *Axiomathes*, vol. 14, no. 1, pp. 57-76, March 2004.