*Proceedings of the 2012 Winter Simulation Conference*
*C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A.M. Uhrmacher, eds*

# CONCEPTUAL SIMULATION MODELING WITH ONTO-UML

## ADVANCED TUTORIAL

Giancarlo Guizzardi

Federal University of Espírito Santo (UFES)
Computer Science Department
Av. Fernando Ferrari
s/n29060-970 Vitória, Espírito Santo, BRAZIL

Gerd Wagner

Brandenburg University of Technology
Institute of Informatics
P. O. Box 101344
03013 Cottbus, GERMANY

## ABSTRACT

Conceptual modeling is of great importance not only to Information Systems and Software Engineering, but also to Simulation Engineering. It is concerned with identifying, analyzing and describing the essential concepts and constraints of a real-world domain with the help of a (diagrammatic) modeling language that is based on a set of basic modeling concepts (forming a metamodel). In this tutorial, we introduce the ontologically well-founded conceptual modeling language *OntoUML* and show how to use it for making conceptual simulation models as the basis of model-driven simulation engineering.

## 1 INTRODUCTION

Even though there is a common agreement that conceptual modeling is an important first step in a simulation engineering project, at the same time it is thought to be the least understood part of simulation engineering (Tako et al. 2010). In a recent panel discussion on conceptual modeling in simulation (Zee et al. 2010), the participants agreed that there is a lack of "standards, on procedures, notation, and model qualities".

In Section 2, we therefore first resort to the disciplines of *Information Systems and Software Engineering (IS&SE)* for better understanding the possible purposes, languages and techniques of conceptual modeling and for answering the question of what is a *conceptual model* (CM). Essentially, conceptual modeling is an activity performed in the analysis phase of a software or simulation engineering project. Its main purpose is to capture, as faithfully as possible, a relevant part of the real-world domain under consideration, using a well-defined (typically diagrammatic) modeling language for making a CM in the form of a digital artifact.

This tutorial is based on our previous research on ontological foundations of conceptual modeling, reported in (Guizzardi et al. 2003, Guizzardi 2005, Guizzardi and Halpin 2008, Guizzardi and Wagner 2010a, Guizzardi and Wagner 2010b, Guizzardi and Wagner 2011a, Guizzardi and Wagner 2011b, Guizzardi 2011).

The main benefit obtained from establishing the ontological foundations of the core concepts of a conceptual modeling language is a clarification of its real world semantics. A clearly defined semantics of the conceptual model of a domain leads to a higher overall quality of the simulation software program built upon that model with respect to comprehensibility, maintainability, interoperability and evolvability.

**978-1-4673-4781-5/12/$31.00 ©2012 IEEE**

## 2 MODEL-DRIVEN SOFTWARE AND SIMULATION ENGINEERING

*Model-driven Engineering* (MDE), also called *model-driven development*, is a well-established paradigm in IS&SE, see, e.g., the *Model-Driven Architecture* proposal of the Object Management Group (MDA 2012). Since simulation engineering can be viewed as a special case of software engineering, it is natural to apply the ideas of MDE also to simulation engineering. There have been several proposals of using an MDE approach in Modeling and Simulation (M&S), see, e.g., the overview given in (Cetinkaya and Verbraeck 2011).

### 2.1 Models in Software Engineering and Information Systems Engineering

Historically, research in conceptual modeling has first been carried out in the computer science field of Database Systems. It started with two proposals for a conceptual data modeling language: the *semantic model* proposed by (Abrial 1974) and the *entity-relationship (ER) model* proposed by (Chen 1976), which triggered the series of ER conferences (ER 2012) starting in 1979. Later it was noticed that conceptual modeling, e.g., in the forms of *enterprise modeling* and *business process modeling*, plays an important role in software engineering, in general.

In MDE there is a clear distinction between three kinds of models as engineering artifacts resulting from corresponding activities in the analysis, design an implementation phases:

1. ***domain models*** (also called ‘computation-independent’ models);
2. platform-independent ***design models***
3. platform-specific ***implementation models***.

Domain models are solution-independent descriptions of a problem domain produced in the analysis phase of a software engineering project. The term ‘domain model’ is synonymous with the term ‘conceptual model’. A domain model may include both descriptions of the domain’s state structure (in conceptual information models) and descriptions of its processes (in conceptual process models). They are solution-independent, or ‘computation-independent’, in the sense that they are not concerned with making any system design choices or with other computational issues. Rather, they focus on the perspective and language of the subject matter experts for the domain under consideration.

In the design phase, first a platform-independent design model, as a general computational solution, is developed on the basis of the domain model. The same domain model can potentially be used to produce a number of (even radically) different design models. Then, by taking into consideration a number of implementation issues ranging from architectural styles, nonfunctional quality criteria to be maximized (e.g., performance, adaptability) and target technology platforms, one or more platform-specific implementation models are derived from the design model.

In the implementation phase, an implementation model is encoded in the programming language of the target platform. Finally, after testing and debugging, the implemented solution is then deployed in a target environment.

A model for a software (or information) system, which may be called a ‘software system model’, does not consist of just one model diagram including all viewpoints or aspects of the system to be developed (or to be documented). Rather it consist of a set of models, one (or more) for each viewpoint. The two most important viewpoints, crosscutting all three modeling levels: domain, design and implementation, are

1. ***information modeling***, which is concerned with the state structure of the domain;
2. ***process modeling***, which is concerned with the dynamics of the domain.

In the computer science field of database engineering, which is only concerned with information modeling, domain information models have been called ‘conceptual models’, information design models have been called ‘logical design models’, and database implementation models have been called ‘physical design models’.

Examples of widely used languages for information modeling are *Entity Relationship Diagrams* and *UML Class Diagrams*, which subsume the former. Examples of widely used languages for process modeling are *(Colored) Petri Nets*, *UML Activity Diagrams* and the *Business Process Modeling Notation (BPMN)*. Some modeling languages, such as UML Class Diagrams and BPMN, can be used on all three modeling levels in the form of tailored variants. Other languages have been designed for being used on one or two of these three levels only. E.g. Petri Nets cannot be used for conceptual process modeling, since they lack the required expressivity.

We illustrate the distinction between the three modeling levels with an example in Figure 1. In a simple conceptual information model of a person, expressed as a UML class diagram, we require that any person has exactly one mother and one father (according to our understanding of reality), expressed by corresponding binary many-to-one associations, and we do not care about the data types of attributes, which is a computational issue, while we do care about the data types of attributes in the design model where we also make the design decision that it is not required that information about the father or mother of a person is available. Finally, in the Java implementation model, we specify Java-specific data types for attributes and we express the binary associations *mother* and *father* with corresponding reference properties.

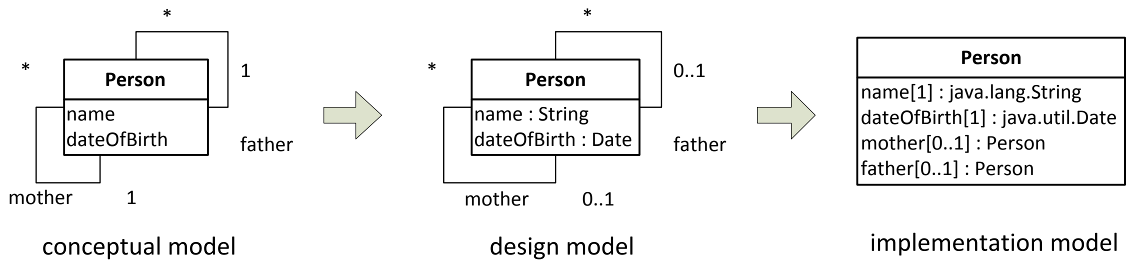

Figure 1: From a conceptual model via a design model to an implementation model of persons.

The fact that the mother/father associations in Figure 1 are mandatory in the conceptual model, while they are optional in the design model, shows that the conceptual model is concerned with the real world, i.e. it takes an *ontological* perspective, while the design model is concerned with the representation of information about the real world, taking an *epistemological* perspective.

## 2.2 Models in Simulation Engineering

Unlike in IS&SE, there is no agreed upon definition, or common understanding, of what is a CM in M&S. Unfortunately, the results achieved in the conceptual modeling field of IS&SE are often ignored by M&S researchers.

A recent panel discussion (Zee et al. 2010) revealed that there are at least three different definitions of what is a CM in M&S:

1. ***a document that states*** "***what you will and will not include in the simulation and why***", or "a repository of high-level conceptual constructs and knowledge specified in a variety of communicative forms (e.g., animation, audio, chart, diagram, drawing, equation, graph, image, text, and video)" intended to assist in the design of a simulation, as proposed by (Balci et al. 2008);
2. "***a formal specification of a conceptualization***", or "an ontological representation of the simulation that implements it" as proposed in (Turnitsa et al. 2010), corresponding to what is called a *domain model* in MDE;
3. "***the specification of an executable simulation model***", or "a non-software specific description of the computer simulation model" as proposed by (Robinson 2008), corresponding to what is called a *design model* in MDE;

Definition 1 reflects a view that is widespread in the M&S community, according to which conceptual modeling is not a well-defined activity resulting in one or more model diagrams expressed in conceptual modeling languages with a well-defined semantics, but rather a loosely defined term referring to all the activities that precede the implementation of a simulation model. In particular, in this view, the issue of information modeling is typically neglected, and only process models are used, often in the form of ad-hoc flow diagrams that are not expressed in a well-defined language. This approach is exemplified by (Ingalls 2008), where entity types are only discussed, but not modeled (e.g., in an Entity Relationship Diagram), and only an ad-hoc process model (a "logic flow" diagram) is presented.

Definition 2 comes closest to the view taken in this tutorial, while definition 3 seems to presuppose that there is nothing like a domain model and the modeling process starts right away with design modeling.

In the MDE approaches of (McGinnis and Ustun 2009) and (Cetinkaya et al. 2011), it is proposed that the CM is to be transformed to a design model or to a simulation program. However, it should be clear from the nature of a CM as a solution-independent description of a domain, that a CM cannot be automatically transformed into a computational specification without human assistance.

Model-driven simulation engineering is based on the same kinds of models as model-driven software engineering: going from a *domain model* via a *simulation design model* to a *simulation implementation model* for the simulation platform of choice (or to several implementation models if there are several target simulation platforms). The specific concerns of simulation engineering, like, e.g., the concern to capture certain parts of the overall system dynamics with the help of random variables, do not affect the applicability of MDE principles. However, they may affect the modeling languages to be used.

We disagree with (Robinson 2011) who states that conceptual modeling "is not a science, but an art" suggesting to make a conceptual model in the form of a set of documents about the M&S project objectives, requirements and design assumptions, completely ignoring the results achieved in IS&SE. Rather, conceptual modeling, both in software and simulation engineering, should be considered an engineering discipline based on scientific research results and best practices.

## 2.3 Conceptual Modeling Languages

There are *general purpose* (domain-independent) modeling languages and domain- specific modeling languages. In the sequel, we simply say 'modeling language' instead of 'general purpose modeling language'.

Discrete event simulation (DES) is concerned with the simulation of real-world systems that are conceived as discrete event systems (or 'discrete dynamic systems'). Such conceptualizations of discrete systems are immaterial entities that only exist in the mind of the user (or a community of users) of a language. In order to be documented, communicated and analyzed they must be captured, i.e. represented in the form of a concrete artifact with the help of a language. The representation of a conceptualization in a language is called a *model* and the language used for its creation is called a *modeling language*. Notice that a discrete event system is placed within a real-world domain, which can be viewed as a *system of systems*. A domain model may, therefore, also be called a *conceptual system model*.

One of the main success factors of a conceptual modeling language lies in its ability to provide a set of modeling constructs that enable its users to directly express relevant domain concepts in an unambiguous manner. A conceptual (or domain) modeling language is a representation of a meta-conceptualization of (a viewpoint of) the real world. We could also say that the meta-conceptualization, which exists in the mind of the language designer, provides an interpretation of the domain modeling language.

A domain (or system) conceptualization, which exists in the mind of the modeler and contains a number of domain (or system) concepts, instantiates a meta-conceptualization. It is represented in the form of a domain (or conceptual system) model expressed in the conceptual modeling language representing the meta-conceptualization. This is illustrated by the diagram shown in Figure 2.

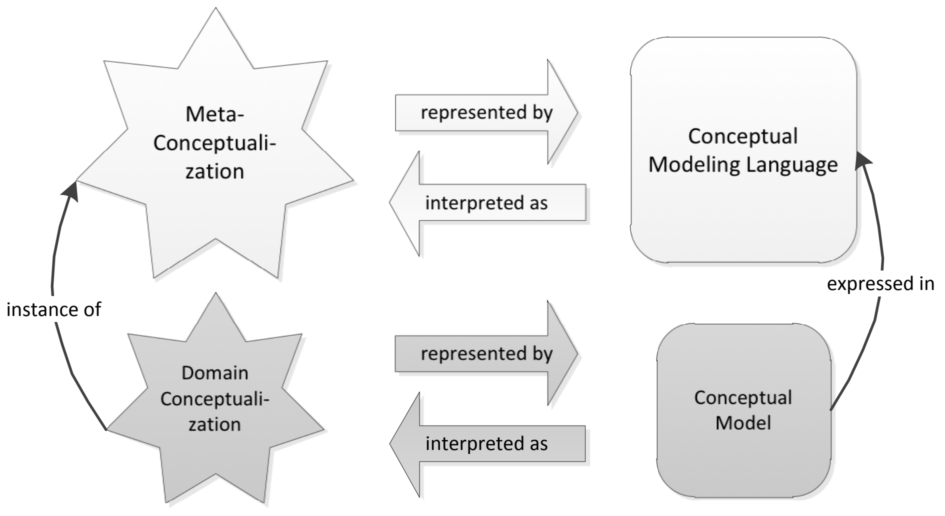

Figure 2: A conceptual modeling language is a representation of a meta-conceptualization.

For instance, for the simple conceptual model of a person shown on the left side of Figure 1, which is a representation of a corresponding domain conceptualization of persons, a subset of UML class diagrams containing language elements for the meta-concepts of classes, attributes and binary associations only, is sufficient as a conceptual modeling language.

Due to their great expressivity and their wide adoption as modeling standards, UML Class Diagrams and BPMN seem to be the best choices for conceptual information and process modeling. However, since they have not been specifically designed for this purpose, we may have to restrict, modify and extend them in a suitable way. In fact, both an analysis of UML with respect to its suitability for conceptual modeling in (Guizzardi 2005) and an analysis of BPMN with respect to its suitability for agent-based DES modeling in (Guizzardi and Wagner 2011) have revealed a number of ambiguities and shortcomings that will have to be resolved and fixed for making these languages fit for conceptual modeling. This issue is further discussed in the next section.

Several authors, e.g. (Wagner et al. 2009), (Cetinkaya et al. 2011) and (Onggo and Karpat 2011), have proposed to use BPMN for discrete event simulation modeling and for agent-based modeling.

## 2.4 Section Summary

- It is natural to apply the general methodology of ***Model-Driven Engineering*** (MDE) also to simulation engineering.
- Unfortunately, the results obtained in the conceptual modeling field of IS&SE have largely been ignored in M&S. Still today, conceptual modeling is often confused with design modeling in M&S.
- A ***conceptual model*** is a solution-independent descriptions of a problem domain expressed in a well-defined (diagrammatic) ***modeling language***.
- In model-driven simulation engineering, we first make a ***conceptual system model***, from which we derive a (platform-independent) ***simulation design model***, which is then transformed into one or more (platform-specific) ***simulation models***.
- A conceptual modeling language is a representation of a ***meta-conceptualization*** of a viewpoint of the real world. A ***system conceptualization***, which exists in the mind of the modeler and contains a number of system concepts, instantiates a meta-conceptualization. It is represented in the form of a conceptual system model expressed in the conceptual modeling language representing the meta-conceptualization.

- We propose to use suitable variants of ***UML class modeling*** and ***BPMN***, which are based on a foundational ontology, for conceptual information and process modeling.

## 3 ONTOLOGICAL FOUNDATIONS OF CONCEPTUAL SIMULATION MODELING

In the DES literature, it is often stated that DES is based on the concept of "entities flowing through the system". E.g., this is the paradigm of an entire class of simulation software such as ARENA (ARENA 2012). However, the loose metaphor of a "flow" only applies to entities of certain types: events, messages, and physical objects may, in some sense, flow, while many entities of other types, such as buildings or organizations, do not flow in any sense. Also, subsuming these three different kinds of flows under one common term "entity flow" obscures their meanings. It is therefore highly questionable, to associate DES with a "flow of entities". Rather, one may say that DES is based on a flow of events.

A discrete event system (or discrete dynamic system) consists of:

- ***objects*** (of various types) whose states may be changed by
- ***events*** (of various types) occurring at times from a discrete set of time points.

For modeling a discrete event system, we have to do the following:

1. describe its ***object types*** and ***event types***;
2. for any event type, specify the ***state changes*** of objects and the ***follow-up events*** caused by the occurrence of an event of that type.

In *Ontology*, which is the philosophical study of what there is, the following fundamental distinctions are made:

- there are *entities* (or *individuals*) and *entity types*, which are called 'universals' in philosophy;
- there are the following categories of entities: *objects*, *trope individuals* (existentially dependent entities such as qualities and relationships) and *events*.

We have discussed these ontological distinctions in depth in (Guizzardi 2005) and in (Guizzardi and Wagner 2010a), where we present our proposal of a *Unified Foundational Ontology* (UFO 2012) based on theories from Formal Ontology, Cognitive Psychology, Linguistics, Philosophy of Language and Philosophical Logics. In (Guizzardi and Wagner 2010b, Guizzardi and Wagner 2011a) we propose the discrete event system ontology DESO and its agent-based extension ABDESO, which are foundational ontologies based on UFO and tailored to the domain of (agent-based) DES.

For pragmatic reasons, we use the term 'object' ambiguously, both for objects in the narrow sense of Aristotelian substances, which are existentially independent entities that are founded on matter and may therefore be better called *physical objects*, and also for other kinds of objects in a broader sense, such as books and customer orders.

In the meta-model shown in Fig 3 we summarize the ontological type categories of DESO that form the foundation of conceptual simulation modeling languages. Entity types classify entities, which are said to be their instances. An entity type may be the domain of attributes and reference properties, which are also entity types since their instances, attributions and references, are entities (in fact, they are trope individuals). The range of an attribute is a datatype, which is an abstract thing (namely a structure consisting of a symbol set as the datatype's *lexical space*, an abstract set as its *value space* and a mapping from the lexical space to the value space). The range of a reference property is an entity type. Reference properties are (binary) relationship types. Since objects may participate in events, an event type may have a number of object types as participant types. Causal laws define the dynamics of a discrete event system (formally, they may be viewed as transition functions). Each causal law has one type of triggering event and zero or more types of resulting events.

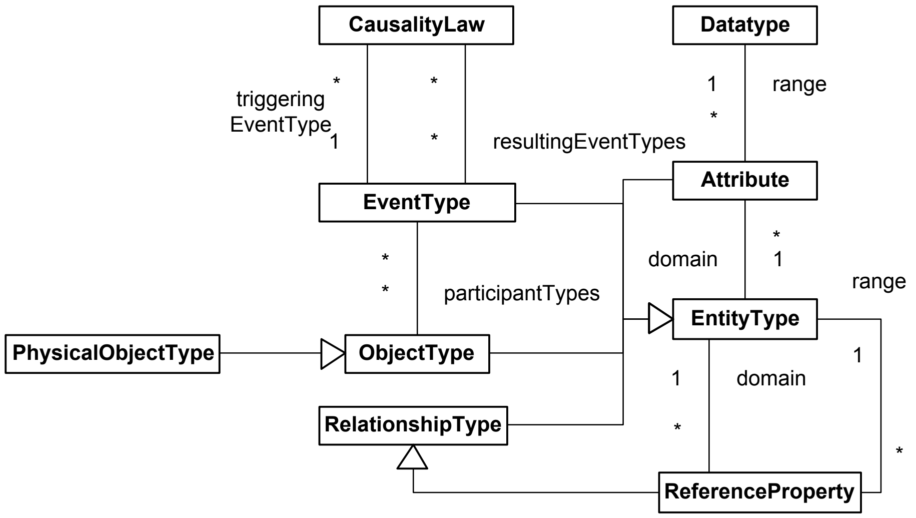

Figure 3: A meta-model describing the basic type concepts of DESO.

## 3.1 Different Categories of Object Types

Any object type is endowed with an *application condition* that allows us to use the object type for classifying objects, that is, to judge if an object is an instance of it. For being able to understand the issues of identity and dynamic classification, we need to make a number of distinctions between different categories of object types. We summarize the theory of object type categories presented in chapter 4 of (Guizzardi 2005).

### 3.1.1 Sortal Types and Mixin Types

A ***sortal type*** is an object type that is endowed with an *object identity condition* for its instances allowing us to judge if two of its instances are the same. The object types Person, Car, Dog, Child and Student are examples of sortal types.

Object types that are not sortal types are called ***mixin types***. Examples of mixin types are RedThing and InsurableItem, as these object types do not provide any identity conditions for their instances, so we could not tell, for instance, if two red objects perceived at different times are the same or not.

We have the following two postulates about mixin types:

- ***A mixin type cannot have direct instances***. This means that a mixin type M must have sortal subtypes, which are directly instantiated by the instances of M.
- ***A mixin type cannot be a subtype of a sortal type***. This is a consequence of the fact that all subtypes of a sortal type are again sortal types since they inherit its object identity condition.

### 3.1.2 Kinds and Subkinds

We define the modal notions of rigidity and non-rigidity for being able to distinguish different categories of object types. An object type O is ***rigid*** if all instances of O are necessarily instances of O (as long as they exist). In other words, if x instantiates O in some possible world, then x must instantiate O in all some possible worlds, in which x exists.

A rigid sortal type may have rigid subtypes, which inherit its object identity condition. A top node in such a rigid sortal type hierarchy is called a ***kind***, while a rigid subtype of a kind is called a ***subkind***,

An important postulate of UFO is:

- ***Every object must instantiate exactly one kind***.

Examples of kinds are Planet, Person and Organization. Examples of subkinds are FemalePerson, which is a rigid subtype of Person, and University, which is a rigid subtype of Organization.

### 3.1.3 Role Types and Phase Types

An object type O is ***anti-rigid*** if no instance of O is necessarily an instance of O. In other words, if x instantiates O in some possible world, then there is another possible world, in which x exists, but does not instantiate O.

An anti-rigid sortal type is a subtype of a kind or a subkind and may be either a *phase type* or a *role type*. In the case of a ***phase type*** P, the specialization condition only depends on attributes of P. For instance, the phase type *Child* classifies persons in a certain age. For a ***role type*** R, in contrast, the specialization condition depends on a relationship type involving R and one or more other sortal types that participate in this relationship type. For instance, the role type *Student* classifies persons who are enrolled in a school. Here, the relationship type is 'enrolled in' and the other involved sortal type is *School*.

In the special case of an anti-rigid mixin type that is partitioned into role subtypes, we speak of a ***role mixin***.

As a direct consequence of the above definitions, we obtain the following postulate:

- ***A rigid object type cannot be a subtype of an anti-rigid object type***.

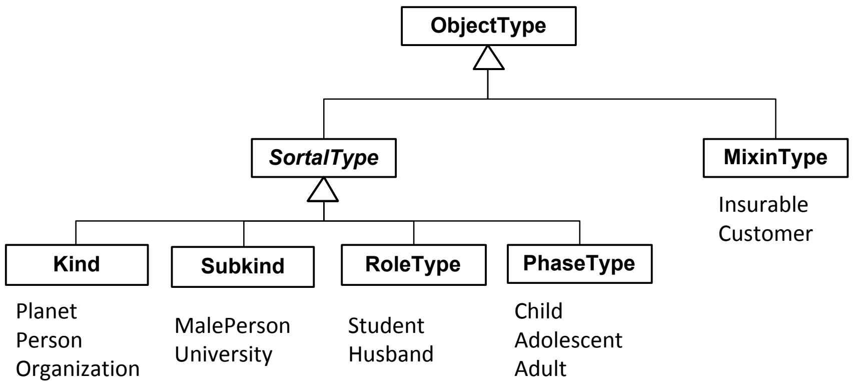

Figure 4: Different categories of object types with instances.

## 3.2 Attributes

We distinguish between the particular color of a particular apple (as a *quality* of the apple) and the color *data value* associated with this particular quality in an ***attribution*** (instantiating an *attribute*). This data value is a member of the *value space* of the *data type* of the attribute. As an example, consider the attribute *hairColor*, the domain of which is the object type Person, and the range of which is a data type with a lexical space consisting of color names. Then, the triple <john, hairColor, grey> represents an attribution that makes the sentence "The hair color of John is grey" true.

A ***quality universal*** classifies individual qualities of the same type. A quality universal can be associated with one or more data types, such that any particular quality corresponds to a specific data value from the value space of the data type. The association between qualities from some quality universal and the corresponding data values from an associated data type is provided by an ***attribute***, which is a universal that classifies *attributions*. A quality universal can be captured by one or more corresponding attributes, each of them based on a different data type.

E.g., the quality universal "hair color" could be captured by an attribute with the range of RGB byte triples or by an attribute with the range of natural language color names. Consequently, we may have more than one attribution for a particular quality, one for each associated attribute.

## 3.3 Relationship Types and Reference Properties

While a *formal relationship*, such as [Brandenburg is part of Germany], holds directly, for a ***material relationship***, such as [Paul is being treated in the medical unit M], to exist, something else, which *mediates* the involved individuals (Paul and M), must exist. Such a mediating individual with the power of connecting individuals is called a *relator*. For example, a medical treatment connects a patient with a medical unit; an enrollment connects a student with an educational institution; a covalent bond connects two atoms. In general, relators are founded on events. A relator universal classifies individual relators of the same type. In the sequel, we simply say 'relationship (type)' instead of 'material relationship (type)'.

Any ***relationship type*** *R* is derived from an underlying relator universal ***R*** that is founded on some type of events. Since each relationship corresponds to a tuple, *R* also has a *tuple extension* (i.e., a *relation* in the sense of set theory). A relationship type is an entity type that classifies relationships, which are 'truth makers' for relationship statements.

A ***reference property*** represents a binary relationship type. Its tuple extension is a subset of the Cartesian product of the extensions of the two involved types. The first type is called the ***domain***, and the second one the ***range*** of the reference property.

## 3.4 The Conceptual Modeling Language OntoUML

We use the UML extension mechanism of a *UML profile* for defining a conceptual modeling language whose elements represent the DESO type categories, including the different categories of object types discussed in Section 3.1 to 3.4. It is important to emphasize, however, that the language defined does not depend on UML, which is used here only for exploiting the convenience of its built-in profile extension mechanism and due to its wide adoption in computer science and its practical relevance. Alternatively, we could have proposed a new modeling language based on the same concepts.

The proposed *OntoUML profile* contains a set of stereotyped classes that support the design of ontologically well-founded conceptual models according to UFO and (AB)DESO. Moreover, the profile also contains a number of constraints that are derived from the postulates stated above restricting the way the modeling elements can be related.

Almost all of the basic type concepts of DESO shown in Figure 3 are directly supported by UML Class Diagrams, where relationship types are called 'associations', as shown by the mapping in Table 1. Only the three categories of *physical object types*, *event types* and *causal laws* are not supported. Consequently, we have to add them to the OntoUML profile in the form of the class stereotypes «physical object type», «event type» and «causal laws».

The postulates of UFO lead to the following OntoUML modeling guidelines:

- Any subkind must be a subtype of a kind.
- Any anti-rigid sortal type must be a subtype of a kind.
- A kind must not be a subtype of a subkind, a phase, a role or a role mixin.
- A subkind must not be a subtype of a phase, a role or a role mixin.
- Mixin types must be represented as abstract classes (which are rendered in UML with class names in italics).
- A mixin type must not be a subtype of a kind, a subkind, a phase or a role.

Table 1: Mapping DESO type concepts to corresponding UML elements

| *DESO type concept* | *Corresponding UML element* |
|---|---|
| Entity type | Class |
| Attribute | Property where type is a datatype |
| Reference property | Property where type is a class stereotyped «object type» |
| Relationship type | Association |
| Object type | Class stereotyped «object type» |
| Physical object type | Class stereotyped «physical object type» |
| Datatype | Datatype |
| Kind | Class stereotyped «kind» |
| Subkind | Class stereotyped «subkind» |
| Role type | Class stereotyped «role» |
| Phase type | Class stereotyped «phase» |
| Mixin type | Class stereotyped «mixin» |
| Role mixin | Class stereotyped «role mixin» |
| Event type | Class stereotyped «event type» |
| Causal law | Class stereotyped «causal law» |

## 4 MAKING CONCEPTUAL INFORMATION MODELS WITH ONTO-UML

We would like to point out that the particular object types chosen to exemplify the proposed type categories are used for illustration purposes only. For example, when categorizing the object type Person as a kind, we are not advocating that Person must be, in general, considered as a kind in conceptual modeling. Rather , the intention is to make the consequences of such a modeling choice explicit. The choice itself, however, is always left to the model designer.

### 4.1 Modeling Types, Not Individuals

When we model a particular discrete system, including the many different things that make up the system, do we model these particular things and this particular system or do we model this type of system with all the types of things that make up such a type of system? The answer to this question seems to vary from case to case. In the case of modeling a machine, it is clear that we are interested in a model of this type of machine, and not in a model of a particular machine. But in the case of modeling an organization, it seems that we want a model of this particular organization only, and we are not really interested in considering the more general case of the type of organization that is instantiated by this particular organization. However, we argue that we should always model types, and not individuals. Even in the case of modeling a particular organization $o$, we should adopt the view that there is not only $o$, but there is also a corresponding type of organization $O$, which is instantiated by $o$, and the goal of our modeling project is to model $O$, and not $o$.

### 4.2 Example: A Service Queue System

In the service queue system example, as implemented in the Simurena Library (Simurena 2012), customers arrive at random times at a service desk where they have to wait in a queue when the service desk is busy. Otherwise, when the service desk is not busy, they are immediately served by the service clerk. Whenever a service is completed, the next customer from the queue will be served, if there is any. A naïve conceptual information model of this system may look as shown in Figure 5. There are one-to-one binary relationship types between the object types ServiceDesk and ServiceClerk and between ServiceDesk and ServiceQueue. The fact that a service queue is composed of zero or more private customers is modeled with the help of the UML composition relationship (rendered with a solid 'diamond' at the side of the aggregate type).

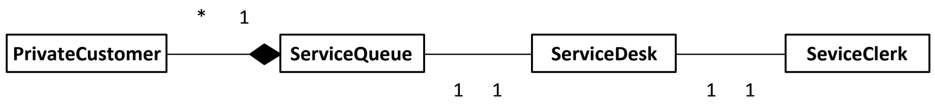

Figure 5: A naïve information model of a service queue system.

Typically, in a simulation design model we would make several simplifications and, for instance, abstract away from the object type ServiceClerk, but in a conceptual system model, we include all entity types that are relevant for understanding the real-world system, independently from the simplifications we make in the solution design and implementation models.

The OntoUML modeling guidelines require that we identify, which object types are kinds, role types or phase types, and that we identify suitable kinds to be added as supertypes of role types and phase types, for making the conceptual model ontologically complete. As a result of following these guidelines we obtain the improved model shown in Figure 6. One of the improvements achieved is that a service clerk may now also be a customer, which is a special case of the real-world possibility that an employee may also be a customer.

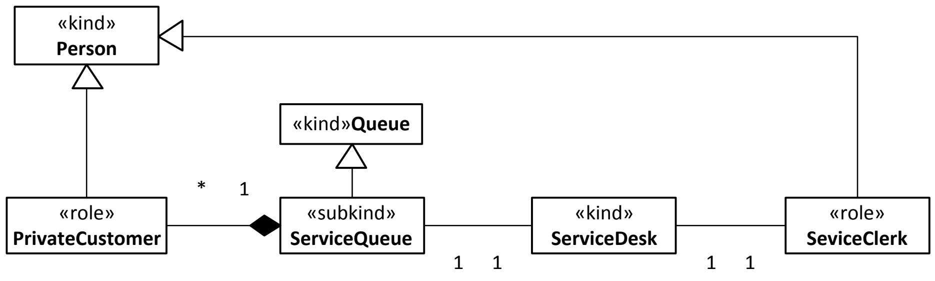

Figure 6: An improved version of the model of Figure 5.

The model of Figure 6 can be further improved by adding event types and related causal laws, as shown in Figure 7. There are three types of events in this system: customer arrival events, service start events and service end events (also called 'customer departure events').

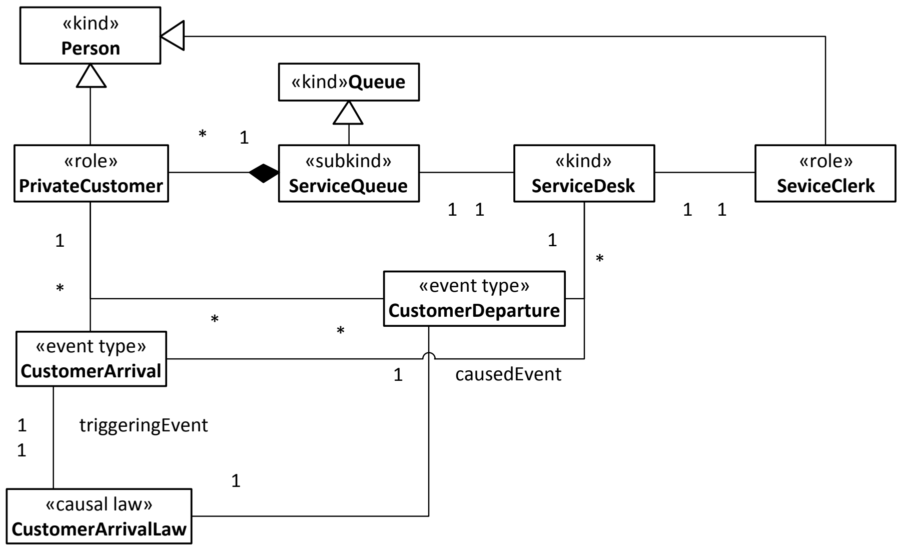

Figure 7: Adding event types and causal laws.

For simplicity, we have omitted service start events, since they can be modeled to coincide either with customer arrival events, when the service queue is empty, or with customer departure events, when the queue is non-empty. In the extended model shown in Figure 7, both customer arrival events and customer departure events have exactly two participants: a customer and the service desk. The CustomerArrival-Law is triggered by a customer arrival event and causes a corresponding customer departure event. Similarly, the CustomerDepartureLaw, which is not shown in the diagram of Figure 7, is triggered by a customer departure event and causes another customer departure event (for the next customer), if the queue is non-empty, as shown in Figure8.

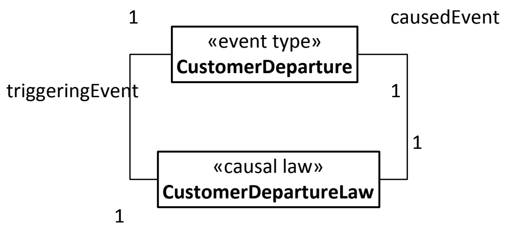

Figure 8: The customer departure law is triggered by a customer departure event.

### 4.3 Example: A Drive Thru Restaurant

In the Drive Thru example, as presented in (Ingalls 2008) and implemented in the Simurena Library (Simurena 2012), cars enter a drive thru from the street and the drivers decide whether or not to get in line. If the driver decides to leave the restaurant, he counts as a lost customer. If he decides to get in line, he waits until the menu board is available. At that time, he gives the order to the order taker at the menu board. After the order is taken, two things occur simultaneously:

1. The driver moves forward if there is room, otherwise he has to wait at the menu board until there is room to move forward.
2. The order is sent back to the kitchen where it is prepared with some delay.

As soon as the driver reaches the pickup window, he pays and picks up his food, if it is ready. If the food is not yet ready, he has to wait until his order is delivered to the pickup window. In (Ingalls 2008), neither a conceptual information model nor an information design model is presented. The only model presented is a “logic flow” model expressed in a diagram language without a clearly defined semantics. Our conceptual information model shown in Figure 9 contains the DriveThru as a subkind of Restaurant, which is a subkind of the kind Organization. The role types OrderTaker, Cook and  PickupWindowClerk specialize the role type Employee, which is a subtype of the kind Person.

## 5 MAKING CONCEPTUAL PROCESS MODELS WITH ONTO-UML

Conceptual process models are based on the event types and causal laws modeled in a conceptual information model underlying the process model. They are expressed in BPMN where a causal law takes the form of an ***event subprocess***, which describes a process type the instances of which are triggered by events of a certain type.

### 5.1 Example: A Service Queue System

The conceptual process model for the service queue system consists of two event subprocesses, one for the CustomerArrivalLaw and one for the CustomerDepartureLaw, as shown in Figure 10. The object type ServiceDesk, the instances of which participate in customer arrival events and customer departure events, is modeled as a ‘data object’.

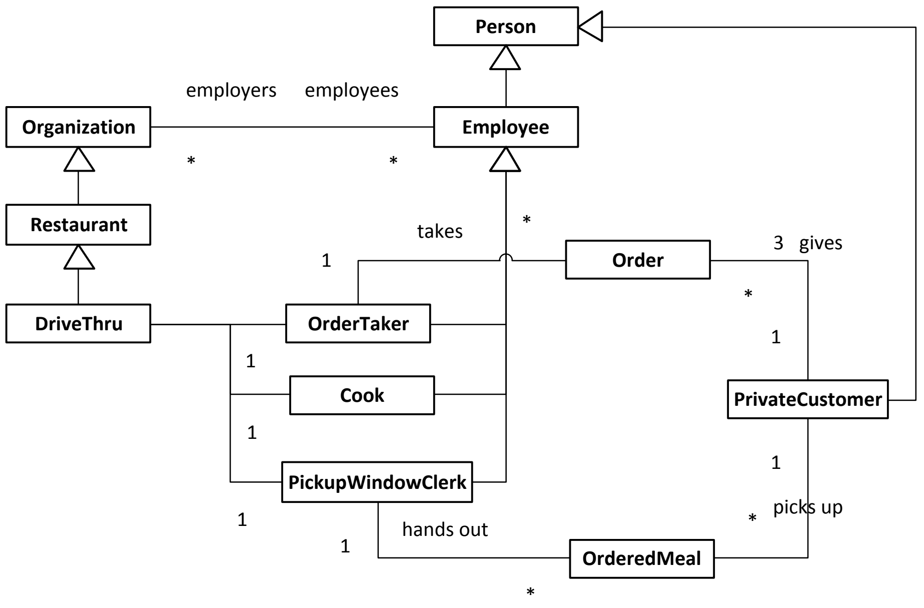

Figure 9: A conceptual model of a drive thru.

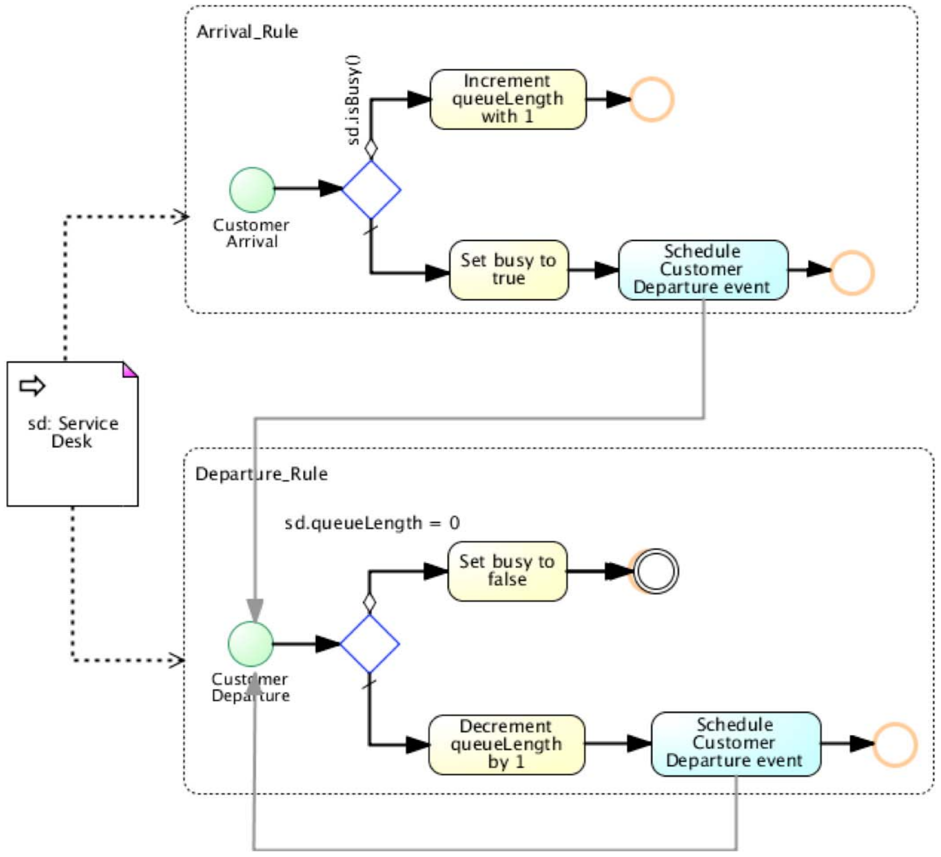

Figure 10: A conceptual process model of the service queue system.

## REFERENCES

Abrial J-R. 1974.'Data semantics. In: Klimbie and Koffeman (eds.), *Data Management Systems*, North-Holland.

ARENA. 2012. http://www.arenasimulation.com. Accessed May 23, 2012.

Balci, O., J.D. Arthur and R.E. Nance. 2008. Accomplishing reuse with a simulation conceptual model. In Proceedings of the 2008 Winter Simulation Conference, ed. S. J. Mason, R. Hill, L. Moench, and O. Rose, 959-965. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Cetinkaya, D. and A. Verbraeck. 2011. Metamodeling and Model Transformations in Modeling and Simulation. In: S. Jain, R.R. Creasey J. Himmelspach, K. P. White, and M. Fu. Proceedings of Winter Simulation Conference. Phoenix, Arizona, USA.

Cetinkaya, D., A. Verbraeck, and M. D. Seck. 2011. MDD4MS: A Model Driven Development Framework for Modeling and Simulation. In Proceedings of the 2011 Summer Computer Simulation Conference (SCSC 2011). The Hague, Netherlands.

Chen P. 1976. The entity-relationship model: Towards a unified view of data, *ACM Transactions on Database Systems* 1(1).

ER (International Conference on Conceptual Modeling) 2012. http://www.conceptualmodeling.org/. Accessed May 23, 2012.

Guizzardi, G., H. Herre and G. Wagner. 2003. On the General Ontological Foundations of Conceptual Modeling", 21st *International Conference on Conceptual Modeling (ER-2002)*. Springer-Verlag, Berlin, Lecture Notes in Computer Science 2503, 65-78.

Guizzardi, G. 2005. *Ontological foundations for structural conceptual models*. PhD thesis, University of Twente, Enschede, The Netherlands. CTIT Ph.D.-thesis series No. 05-74 ISBN 90-75176-81-3.

Guizzardi, G. and T. Halpin (2008). Ontological Foundations for Conceptual Modeling. Applied Ontology, vol. 3, p. 91-110.

Guizzardi, G. and G. Wagner. 2010a. Using the Unified Foundational Ontology (UFO) as a Foundation for General Conceptual Modeling Languages. In Poli R., M . Healy and A. Kameas (Eds.), *Theory and Applications of Ontology: Computer Applications*., 175−196. Available from: http://www.inf.ufes.br/~gguizzardi/TAO-CR.pdf

Guizzardi, G. and G. Wagner. 2010b. Towards an Ontological Foundation of Discrete Event Simulation. In: Johansson B, Jain S, Montoya-Torres J, Hugan J, Yücesan E (Eds.), Proceedings of Winter Simulation Conference, Baltimore (MD), USA, 652−664. Available from: http://www.informs-sim.org/wsc10papers/059.pdf

Guizzardi, G. and G. Wagner. 2011a. Towards an Ontological Foundation of Agent-Based Simulation. In: S. Jain, R.R. Creasey J. Himmelspach, K. P. White, and M. Fu. Proceedings of Winter Simulation Conference. Phoenix, Arizona, USA, 284−295. Available from: http://www.informs-sim.org/wsc11papers/024.pdf

Guizzardi, G. and G. Wagner. 2011b. Can BPMN Be Used for Making Simulation Models? In: J. Barjis, T. Eldabi and A. Gupta (Eds.). *Enterprise and Organisational Modeling and Simulation*. Springer Lecture Notes in Business Information Processing, vol. 88.

Guizzardi, G. 2011. Theoretical Foundations and Engineering Tools for Building Ontologies as Reference Conceptual Models, *Semantic Web Journal*, IOS Press, Amsterdam, 2011.

Ingalls, R.G. 2008. Introduction to Simulation. In Mason, S.J. Hill, R.R. Mönch, L. Rose, O. Jefferson, T. Fowler, J.W. (Eds.), *Proceedings of the 2008 Winter Simulation Conference*, 17–26.

McGinnis, L. and V. Ustun. 2009. A Simple Example of SysML-Driven Simulation. In: M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin and R. G. Ingalls, (Eds.), Proceedings of Winter Simulation Conference.

MDA (Model-Driven Architecture). 2012. http://www.omg.org/mda/. Accessed May 23, 2012.

Onggo, B. S. S. and O. Karpat. 2011. Agent-Based Conceptual Model Representation Using BPMN. In: S. Jain, R.R. Creasey J. Himmelspach, K. P. White, and M. Fu. Proceedings of Winter Simulation Conference. Phoenix, Arizona, USA.

Robinson, S. 2008. Conceptual Modelling for Simulation Part I: Definition and Requirements. *Journal of the Operational Research Society* **59** (3): 278-290.

Robinson, S. 2011. Choosing the Right Model: Conceptual Modeling for Simulation. In: S. Jain, R.R. Creasey J. Himmelspach, K. P. White, and M. Fu. Proceedings of Winter Simulation Conference. Phoenix, Arizona, USA.

Tako, A.A., K. Kotiadis and C. Vasilakis. 2010. A Participative Modelling Framework For Developing Conceptual Models in Healthcare Simulation Studies. In: Johansson B, Jain S, Montoya-Torres J, Hugan J, Yücesan E (Eds.), Proceedings of Winter Simulation Conference, Baltimore (MD), USA, 500−512. Available from: http://www.informs-sim.org/wsc10papers/045.pdf

Simurena Library. 2012. A public library of simulations and games for education and entertainment. Accessed May 23, 2012. http://portal.simulario.de/public/.

Turnitsa, C., J.J. Padilla and A. Tolk. 2010. Ontology for Modeling and Simulation. In: Johansson B, Jain S, Montoya-Torres J, Hugan J, Yücesan E (Eds.), Proceedings of Winter Simulation Conference, Baltimore (MD), USA, 643−651. Available from: http://www.informs-sim.org/wsc10papers/058.pdf

UFO (Unified Foundational Ontology). 2012. http://www.ufo-ontology.info/. Accessed May 23, 2012.

Wagner, G., O. Nicolae and J Werner. 2009. Extending Discrete Event Simulation by Adding an Activity Concept for Business Process Modeling and Simulation. In: M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin and R. G. Ingalls, (Eds.), Proceedings of Winter Simulation Conference, 2951-2962.

Wiggins, D. 2001. *Sameness and Substance Renewed*. Cambridge University Press.

Zee, D.-J. van der, K. Kotiadis, A.A. Tako, M. Pidd, O. Balci, A. Tolk and M. Elder. 2010. Panel Discussion: Education on Conceptual Modeling for Simulation – Challenging the Art. In: Johansson B, Jain S, Montoya-Torres J, Hugan J, Yücesan E (Eds.), Proceedings of Winter Simulation Conference, Baltimore (MD), USA, 290−304. Available from: http://www.informs-sim.org/wsc10papers/026.pdf

## AUTHOR BIOGRAPHIES

**GIANCARLO GUIZZARDI** is Associate Professor at the Computer Science Department, Federal University of Espírito Santo (UFES), Brazil, and senior member of the Ontology and Conceptual Modeling Research Group (NEMO). His work is focused in the development of domain ontologies and foundational ontologies and their application in computer science and, primarily, in the area of conceptual modeling and organizational modeling. He has been involved in a number of industrial projects in domains such as off-shore software development, petroleum and gas, medical informatics, telecommunications and news information management. His email address is gguizzardi@inf.ufes.br.

**GERD WAGNER** is Professor of Internet Technology at the Department of Informatics, Brandenburg University of Technology, Germany. His research interests include agent-oriented modeling and agent-based simulation, foundational ontologies, (business) rule technologies and the Semantic Web. In recent years, he has been focusing his research on the development of an agent-based discrete event simulation framework, called *ER/AOR Simulation* (see www.AOR-Simulation.org) He can be reached at http://www.informatik.tu-cottbus.de/~gwagner/.