OntoUML and UFO-A for Software Engineering

Robert PERGL

Zdeněk RYBOLA

David BUCHTELA

Ivan RYANT

Abstract: OntoUML is an extension to the well-known notation of UML by ontology-oriented conceptual modeling aspects. The OntoUML diagrams offer higher expressivity for conceptual modeling thanks to a finer categorization and definition of entity types. This paper summarizes basic principles and concepts of OntoUML in the perspective of using OntoUML for development of information systems. The advantages of higher expressivity of OntoUML are illustrated by an example. Other aspects like transformation to an implementation model and further development of OntoUML are discussed. Difficulties for wider spread of OntoUML in the professional community are also discussed.

Keywords: ontological modeling, conceptual modeling, OntoUML, software engineering, information system development.

JEL Classification: C63, C80

Introduction

The process of developing complex business information system consists of several consecutive activities (Beck 1998). Assuming the requirements are defined and the project infrastructure is set, three crucial phases lead to the software realization – not considering consecutive phases of testing, deployment and support:

1) Analysis,

2) Design,

3) Implementation.

Figure 1. Simplified model of software development process denoting the key artifacts between the phases

In Figure 1, artifacts are shown to serve as inputs and outputs between phases of the process. The quality of the input artifacts significantly influences the success of each phase – a high quality design cannot be created from low quality analytical sources and a high quality implementation cannot be created from a low quality design. The quality is influenced especially by these two factors:

- The quality of the modeling notation (Guizzardi, Ontological Foundations for Structural Conceptual Models 2005).
- The preservation of information between phases (Pícka and Pergl 2006).

In this paper, we deal especially with the quality of the analytical output artifacts, in particular with the conceptual model. Furthermore, we discuss the transformation of the structural conceptual model to a structural implementation model in the design phase.

Goals

The goal of this paper is to outline the possibilities and advantages of using the OntoUML notation for conceptual modeling of business information systems. We deal only with the structural models of the systems. Another goal of this paper is to outline the issue of transformation of the OntoUML models into implementation models, in particular for the pure object-oriented models. A side goal of the paper is to introduce the OntoUML notation that is not yet well-known by the professional public and the difficulties for the wider spread of this notation.

Structure of the paper

In section 2, we introduce the origin and structure of OntoUML. In section 3, we show the fundamentals and principles of OntoUML and provide the basic categorization of entity types. Higher expressivity of OntoUML is demonstrated on an example by comparing a UML and an OntoUML model. We discuss the task of transformation of an OntoUML conceptual model into an implementation model in section 4. Finally, in section 5, we outline the difficulties for wider usage of OntoUML by professional community and we provide conclusions and further plans.

Origins of OntoUML

OntoUML was created as an attempt to merge the ontological analysis and conceptual modeling. The goal of the intention was to provide the analyst with a set of various entity types with precisely defined qualities and characteristics to model the reality as precise as possible. OntoUML is based on the Cognitive Science knowledge about spe-

cifics of our perception and on the modal logic and the mathematical foundations of logic, sets and relations. Syntactically it is built on the notation of UML class diagrams (Fowler 2003) that is extended by a set of new concepts by special stereotypes – this formally conforms to the UML metamodel (OMG n.d.) and therefore current CASE tools, presentation tools, transformation tools, etc. can be used for OntoUML models. Unlike other extensions of UML, OntoUML is created from the very foundations and constitute a complete system independent of the original UML elements. It uses some aspects (like classes), however, it omits a set of other problematic concepts (for instance aggregation and composition) and replaces them with own ontologically correct concepts.

For the first time, OntoUML was presented in the dissertation thesis by Dr. Giancarlo Guizzardi that was defended with highest honors in 2005 at the University of Twente. The thesis was later published as a monograph (Guizzardi, Ontological Foundations for Structural Conceptual Models 2005). Since then, Dr. Guizzardi is one of the most active scientist and author in the field of conceptual modeling and he is a member of many conference boards in the field [1]. Currently, Dr. Guizzardi works at the Federal University of Espírito Santo in Vittoria, Brasil, where he leads the NEMO research group.

Three foundational ontologies are created using OntoUML – UFO (Unified Foundational Ontology):

1) UFO-A: Structural aspects – Objects, their types, parts, roles they play …

2) UFO-B: Dynamic aspects – Events, their parts and relations, object participation in events, time-dependent behaviour …

3) UFO-C: Social aspects – Based on UFO-A and UFO-B, dealing with agents, states, goals, actions, norms, social commitments and claims …

UFO-A is considered to be completed and proved in a set of projects in practice, no more intensive research for its extension and development is expected. On the other hand, UFO-B and UFO-C are targets of current intensive research.

OntoUML and UFO has already been applied in practice in many complex projects, for example Off-Shore Software Development – conceptual analysis for an international oil company, in projects in the field of telecommunications and Media Content Management.

1) A list of publications available for downloading is published at the personal webpage of Dr. Guizzardi (Guizzardi, Homepage n.d.).

Basic Principles of OntoUML

In the following, we deal only with the UFO-A and the OntoUML term is used for the notation and UFO-A together. We focus on the structural aspects for static modeling of structures.

Categories of the entity types

As mentioned above, OntoUML strictly distinguishes various categories of entities in the reality around us. Part of the entity types' hierarchy is shown in Figure 2.

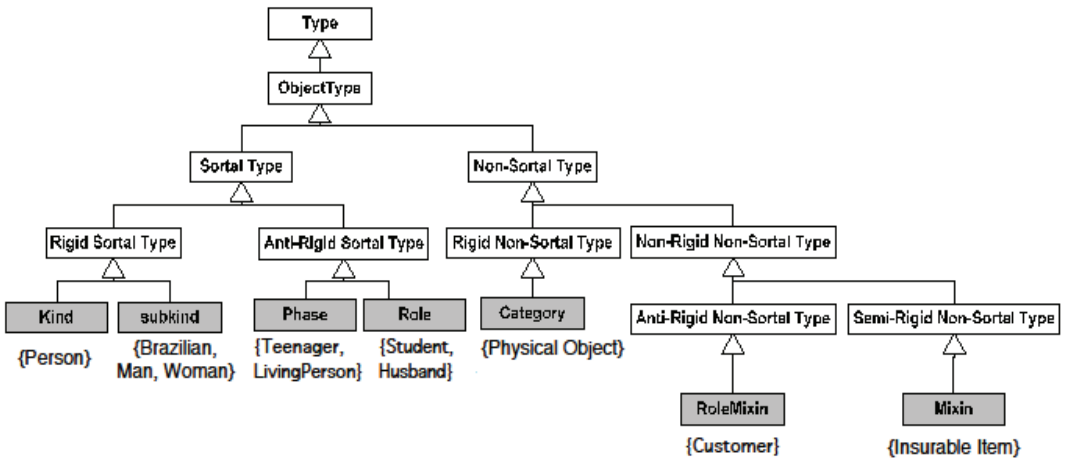

Figure 2. The hierarchy of basic entity type categories in OntoUML

A set of strict criteria is defined to distinguish the categories:

- Identity – if the entity type provides or has an ontological identity. The types that do not provide identity inherit the identity from their ancestors.
- Rigidity – the changeability or immutability of the type. OntoUML uses modal logic for definitions of various aspects using operators necessity and possibility in time and space – modal logic uses the term world. We distinguish rigidity, anti-rigidity and non-rigidity (the negation of rigidity). Based on these characteristics, we distinguish rigid, anti-rigid and semi-rigid – for some instances rigid, for other non-rigid) entity types.
- Relational dependency – the entity type can exist only with a relation to another entity type.

The table shown in Figure 3 summarizes characteristics of the basic entity type categories.

Category of Type	Supply Identity	Identity	Rigidity	Dependence
SORTAL	-	+		
« kind »	+	+	+	-
« subkind »	-	+	+	-
« role »	-	+	-	+
« phase »	-	+	-	-
NON-SORTAL	-	-		
« category »	-	-	+	-
« roleMixin »	-	-	-	+
« mixin »	-	-	~	-

Figure 3. Characteristics of the basic entity type categories

Strict distinguishing of the entity type's category has a practical significance in the software engineering. An example of a UML model and its OntoUML counterpart is shown in Figure 4. It describes a small part of a university information system. The UML model defines only four entity types for a general person, a student, an employee and an insured employee. The OntoUML counterpart strictly distinguishes several various categories of the entity types and adds some other required types. The UML model misses some important information that can lead to faulty design and implementation:

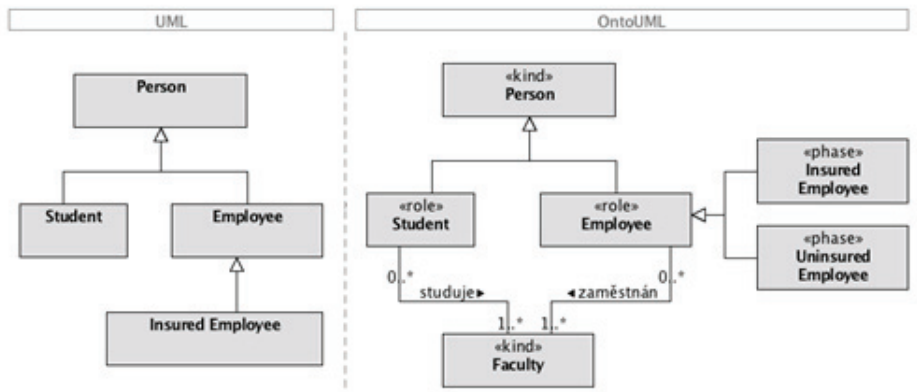

Figure 4. An example of a UML model and its OntoUML counterpart for a university information system

- In the UML model, we distinguish students and employees, both as subtypes of a person. However, a person cannot be a student and an employee in the same time – in this case, two independent instances must be created [2]. On the other hand, in the OntoUML model, we classify the student and employee types as roles of the person type. This means that each person can be in the role of a student or an employee at a faculty. The role category is anti-rigid. This classification enables the person to gain or lose the role or even be in both roles at the same time.

- In the OntoUML model, the roles inherit the identity from the kind Person. This allows a person to gain or lose a role without losing its identity in the system. In the UML model, the student and the employee have their own identity. When a student finishes its studies and becomes an employee, it loses all its history in the system.

- Phases in the OntoUML model are anti-rigid, too. This means that the employee can change its state between the insured employee and the uninsured employee without changing its identity. Unlike the roles, any person can be only in one and exactly one phase of the same phase partition. The rigid UML model does not support changing the insurance state of a person without changing the person's identity. We could create an optional association between an employee and an insurance entity type; however, we would lose the polymorphism of employees. In the OntoUML, we can simply create an association between the insured employee and a car entity type denoting that only an insured employee can drive a car. This constraint cannot be expressed in pure UML [3].

- A role in OntoUML is a relationally dependent type – there must be another entity type connected to the role by an association with the minimal multiplicity value of 1. This rule forces the analyst to search for the truth maker of the role – what makes the kind entity to get the role – and to include it in the model. In UML, there is no such concept.

Characteristics of the Whole-Part relationship

OntoUML also deals with the Whole-Part relationship in more detail than it is defined in UML. UML defines composition and aggregation associations very vaguely and imprecisely. OntoUML brings more precise definition of the relationship and its multiplicities and obligation from the point of view of both the whole and the parts. Omitting these important aspects can result to faulty design and implementation because of the missing information.

2) There exist implementation solutions for such a situation, however, we focus on the structural conceptual modeling and the way to express such characteristics on this level. We try to respect the rule that the conceptual model should be maximal and complete.
3) Such constraints can be expressed by OCL invariants attached to the diagram, however, these invariants can not be expressed directly by the diagram elements.

Types of the obligatory participation from the point of view of the whole entity

The types of the obligatory participation from the perspective of the whole entity are shown in Figure 5.

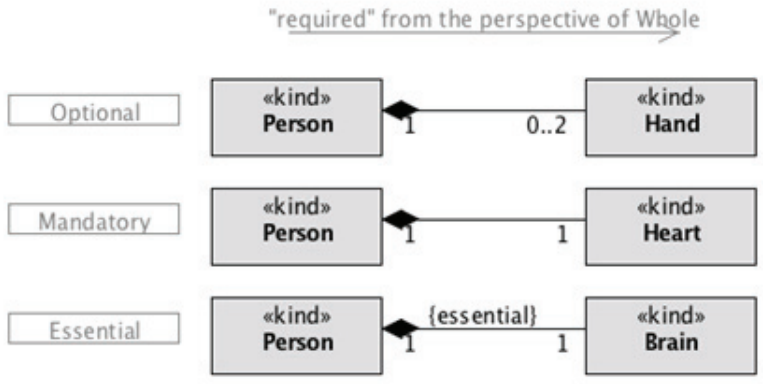

Figure 5. The types of the obligatory participation of the part entity from the perspective of the Whole

- Optional Part – the part entity is optional for the whole entity, the whole instance can exist without any instance of the part entity.

- Mandatory Part – the part entity is obligatory for the whole entity. Any instance of the whole must be linked to an instance of the part entity. However, the part entity instance may change. An example of this type of obligation is a human heart: a human must have a heart to live but the heart can be transplanted – the instance of the heart is changed while the human instance does not. Another example could be an engine of a car.

- Essential Part – the part entity is obligatory for the whole entity. Additionally, the part entity is essential – the part entity instance can not be changed. Changing the part entity instance would destroy the identity of the whole entity instance. An example of this type of obligation is a human brain – the brain can not be transplanted and even if it could be the human would be someone else with new identity. Similarly, the chassis of the car can not be changed without changing the car's identity because the VIN number is printed on it.

Types of the obligatory participation from the point of view of the part entity

The types of the obligatory participation of the whole entity from the perspective of the part entity are shown in Figure 6.

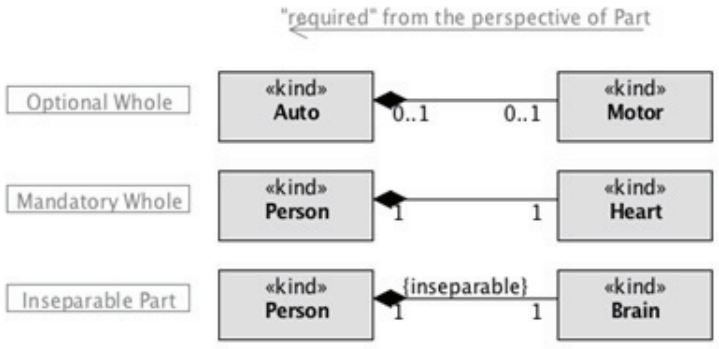

Figure 6. The types of the obligatory participation of the whole entity from the perspective of the part

- Optional Whole – the whole entity is optional for the part entity. An instance of the part entity can exist without any instance of the whole entity.
- Mandatory Whole – the whole entity is obligatory for the part entity. An instance of the part entity must be always connected to an instance of the whole entity. However, it can change the whole entity's instance. An example of this type is a kidney that can be transplanted from one person to another.
- Inseparable Part – an instance of the part entity must be a part of the same whole entity instance for its whole life. An example of this type can be the human brain, again, as it can not be transplanted.

Other characteristics of the participation in the Whole-Part relationship

All types of the obligation mentioned above stand for rigid whole and part entities. When one of the types is anti-rigid, we have to distinguish, if the instance linked to the anti-rigid type instance can change when the type changes its anti-rigid state – remember, a rigid entity can gain or lose a role and change its phase and when it gains one of the roles or phases, it is always linked to the same instance of the other type. This characteristic is expressed by these meta-attributes of the relationship:

- **Immutable Part** – when an instance of the whole entity is in the anti-rigid state, it is always connected to the same instance of the part entity.
- **Immutable Whole** – when an instance of the part entity is in the anti-rigid state, it is always connected to the same instance of the whole entity.

OntoUML also defines the shareability of the part entity instance among the whole entity instances. This characteristic uses similar notation as aggregation and composition in UML but with a different meaning:

- The full symbol ♦ or the meta-attribute isShareable = false stands for a not-shareable part – an instance of the part entity can be a part of only one whole entity instance at the same time.
- The empty symbol ◊ or the meta-attribute isShareable = true stands for a shareable part – an instance of the part entity can be a part of multiple instances of the whole entity at the same time.

It is important to mention that all these characteristics – the types of obligatory participation from the perspective of both the part and the whole entity and the shareability of the part entity instances are completely independent of each other allowing modeling of any combination of these characteristics as required by the domain reality.

Types of the Whole-Part relationship

OntoUML also distinguishes various types of the Whole-Part relationship and the roles the parts take in the whole:

Quantity

- The whole consists of parts of the same type.
- The whole is infinitely dividable into parts.
- The part represents all the material of the container.
- Examples: water in a bottle, stone of a statue, etc.
- Relation subQuantityOf: alcohol-wine, sugar-coffee, etc.

Collective

- The whole consists of parts of other types.
- The collective is not infinitely dividable.
- All parts take the same role in the collective.
- Relation memberOf: a tree – a forest, a student – a class, etc.
- Relation subCollectionOf: north part of a forest – a forest, cars – vehicles, etc.

Functional Entity

- The whole consists of parts that take various roles.
- Relation componentOf: a heart – a vascular system, a director – a company, an engine – a car, etc.

Other parts of the UFO-A

UFO-A also deals in detail with associations, aspects (existentially dependent objects), qualities (complex measurable attributes) and their domains and also with a redefinition of specialization.

Transformation of an OntoUML conceptual model into an implementation model

While the conceptual model is the result in the case of conceptual analysis, in software engineering it is just one of the first input artefacts for other development phases. The process of implementation and source code creation based on the conceptual OntoUML model with a set of additional information and rules not captured in the model (behavioural models, state models, etc.) is possible but error-prone because the semantic gap between the models is huge. The conceptual model also does not contain some additional information required for the implementation just because of the principle of the conceptual model – it must be independent of the implementation. However, the implementation requires information like the associations' direction for object navigation, queries' optimization and structure actualization.

Therefore, we suggest creating an implementation model that captures how the conceptual model will be implemented. This model contains the structural information of the conceptual model along with additional implementation dependent information and it can be transformed into the source code.

Currently, we focus on the pure object-oriented implementation so the implementation model consists of classes, attributes, methods, inheritance and composition. Of course, the conceptual model degenerates during the transformation, therefore even the conceptual model must be discussed during the implementation.

Another option is the transformation to a relational model. The NEMO research group focused on this transformation; however, the results are not public because the research was done as a part of a commercial project. The relational implementation model seems to be more complicated because of the required application logic to make the implementation valid according to the former conceptual model (in PL-SQL, for example).

Conclusions

OntoUML and UFO can be used as tools for ontology-oriented conceptual modelling in many areas where an exact mental model needs to be expressed:

- in the area of information exchange,
- in the area of term and relations definition (legal regulations, etc.),
- in the area of data integration (business and knowledge engineering, business intelligence, e-government, etc.),
- in the area of service and component integration (heterogeneous environment).

In software development, the information exchange is crucial for success of the project and for satisfaction of the customer's requirements and needs. OntoUML and UFO--A provide higher expressivity and precision for structural conceptual model than UML models.

Important aspects of OntoUML

In our experience and opinion, a subset of OntoUML aspects is sufficient for a software engineer. We consider these aspects as the most helpful:

- The distinguishing of various categories of entity types into Sortals and Non-Sortals.
- The distinguishing of rigidity of entity types.
- The distinguishing of the obligatory participation in the Whole-Part relations.
- The distinguishing of various types of the Whole-Part relations.
- The distinguishing of material and formal relations.
- The distinguishing of aspect categories (qualities and modes).

We believe that a software engineer finds the basic definitions and characteristics of the mentioned concepts sufficient for the software development practise. Dr. Guizzardi introduces very deep theory of mereology for the Whole-Part relations issue, however, we do not consider this necessary for the software development.

Difficulties of OntoUML spread in the public community

OntoUML is not yet well-known a spread among the community of software engineers. There are several difficulties that prevent its wider spread. We find the following problems the most important:

- Lack of literature and information sources.
- Lack of public examples and case studies.
- Missing courses and training programs.
- Hardly available support and community.
- Missing CASE tools.

In the following paragraphs, the problems are described in more detail.

Lack of literature and information sources

There is only one complex and publicly available publication about OntoUML – the dissertation thesis of Dr. Guizzardi (Guizzardi, Ontological Foundations for Structural Conceptual Models 2005). It is a top-class research thesis; however, it is really tough and intense for the public community. Additionally, it does not describe the most recent version of OntoUML, one have to study the most recent conference papers, in English.

Lack of public examples and case studies

As mentioned above, OntoUML was applied in several successful projects; however, these projects were commercial and the results are kept private. Some case studies were published in (Guizzardi, Homepage n.d.) but it is too little for adequate education.

Missing courses and training programs

OntoUML is taught by the Dr. Guizzardi's team only at the Federal University of Espírito Santo in Vittoria, Brasil, and at some universities in Holland. However, we started to teach OntoUML at FIT CTU in Prague in the last year. Also, no commercial courses for the public community of software engineers are available worldwide.

Hardly available support and community

The community of OntoUML users and developers is too small and consists almost exclusively of the NEMO team of Dr. Guizzardi and several other researchers in Holland and Germany. The community is very important to support developers new to OntoUML and to provide help, tips and sources for them. Currently, we try to build a community of OntoUML developers at FIT CTU in Prague.

Missing CASE tools

Currently, a CASE tool based on the Eclipse platform created as a part of Dr. Guizzardi dissertation thesis is available for OntoUML modelling. The tool supports all OntoUML concepts of UFO-A and is capable of model checking according to the rules for OntoUML models. However, the tool is not developed anymore and misses many important functions for practical usage in software development. OntoUML models can be also created in many current CASE tools for UML, however, these tools provide no semantic support for OntoUML concepts.

Future plans and perspectives

The NEMO team currently works on a software tool for behavioural simulation of entities in a structural diagram. Such a tool could help a software engineer to verify a conceptual model and validate it with the customer on a prototype of the structures.

Finishing the rules and definitions for the transformation of an OntoUML model into various types of implementation models can provide a powerful tool for rapid development, flexible adaptability of models and consistency of the model and its implementation. However, this transformation will probably require additional information and therefore it can be only semi-automatic.

UFO-B is intensively developed by the NEMO team. When it is developed enough, modelling of dynamic aspects of information systems in OntoUML will be possible. Although many methods, techniques and notations are available for process modelling, the situation is similar to the structural modelling – the notation is not satisfactory enough from the ontological point of view – see the analysis of BPMN notation (Guizzardi, Can bpmn be used for making simulation models? 2011).

Acknowledegement

The OntoUML research, studies and teaching is supported by the Centre for Conceptual Modelling and Ontologies at Faculty of Information Technologies, Czech Technical University.

References

[1] Beck, K. Process Patterns: Building Large-Scale Systems Using Object Technology. Cambridge University Press, 1998.

[2] Fowler, M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd edition edn. Addisson-Wesley Professional, 2003.

[3] Guizzardi, G. Homepage. http://nemo.inf.ufes.br/gguizzardi. —. "Can bpmn be used for making simulation models?" Lecture Notes in Business Information Processing - LNBIP, 2011: 100-115.—. Ontological Foundations for Structural Conceptual Models. Telematica Instituut Fundamental Research Series, 2005.

[4] OMG. Metaobject facility. http://www.omg.org/mof.

[5] Pícka, M., and R. Pergl. "Gradual modeling of information system: Model of method expressed as transitions between concepts." ISAS, 2006: 538-541.

Department of Software Engineering, FIT CTU in Prague, Thákurova 9, 160 00, Praha 6
{robert.pergl, zdenek.rybola, david.buchtela, ivan.ryant}@fit.cvut.cz