

Interoperability Services for Models and Ontologies¹

Jürgen EBERT, Tobias WALTER

University of Koblenz-Landau

Universitätsstr. 1

D-56070 Koblenz

e-mail: {ebert,walter}@uni-koblenz.de

Abstract. Model-based approaches in the UML/MOF technological space and ontology-based approaches in the OWL technological space both support conceptual modeling using different kinds of representation and analysis technologies. Both spaces provide services for accessing and manipulating models and ontologies, respectively.

This paper compares both spaces using to a mapping of models and ontologies based on a common model-theoretic semantics. Based on this mapping, different services for bridging the respective technological spaces are defined and discussed. Three different bridging strategies, namely Adaptation, transformation, and integration, are distinguished.

Introduction

Models are central artifacts in model-driven software engineering. They are created, edited, and transformed during software development, and querying and reasoning is performed on them. The concrete languages, techniques, and tools used to support these activities depend on the environment the software development processes is performed in. Modeling in general can be done in different *technological spaces* [1], e.g., using UML-like modeling in an eclipse-based environment or using ontology-based modeling making extensive use of reasoning facilities.

The heterogeneity of and the insufficient interoperability between technological spaces often hinders the simultaneous use of their respective advantages. In this paper, which is an enhanced version of [2], we tackle the interoperability problem between the UML-inspired world which we call *Modelware* and the ontology-inspired world, called *Ontoware*. We investigate different ways of bridging between those two spaces.

To define these bridges between the spaces, their similarities (commonalities) and their dissimilarities (variabilities) are identified first. These are elaborated by specifying the semantics of the modeling languages of both worlds in a common approach (namely graph-based model-theoretic semantics), which leads to a mapping of the underlying concepts of both. This mapping is used as a basis for building concrete bridges. We

¹This work has been partially funded by the EU (grant number 216691 in the 7th framework programme), see <http://www.most-project.eu>.

discuss the three main bridging approaches and derive a set of services needed to support the respective kinds of interoperability.

Modelware and Ontoware. Models are used to describe those parts of reality that are relevant for a software system to be built (domain models, *descriptive modeling*), and models are also used to prescribe the target system (metamodels, *prescriptive modeling*). Descriptive modeling is also called conceptual modeling in the knowledge representation literature. Conceptual models in particular describe the relevant classes of entities and relationships in the domain of interest. We use the term *conceptual model* as a general term, whereas the term *model* applies to concrete artifacts from the Modelware world and *ontology* pertains to artifacts from the Ontoware world.

With the advent of the Semantic Web, ontologies found their way into software development, as well [3,4]. Though in the stronger sense an ontology is “a formal explicit specification of a shared conceptualization for a domain of interest” [5], the term ontology is today actually being used for logical knowledge bases in a broader sense. Since Semantic Web technology is heavily based on description logics (DL) [6] in particular, the term seems even more and more to be used as a synonym for description-logics-based knowledge bases in general. This DL-based way of modeling is shortly characterized as *Ontoware* in the following, opposed to MOF-like modeling using *Modelware*, where structural models are seen as networks of objects which can easily be implemented in modern object-oriented languages.

Goal of this paper. Common use of the two technological spaces of Modelware and Ontoware and bridging them in particular requires a thorough understanding of the similarities and dissimilarities of both spaces. Thus, to investigate more deeply on this topic in order to really “marry” both worlds, *concrete languages* with concrete syntaxes and semantics have to be considered. On the Modelware side there is a predominance of OMG’s UML/MOF-based approaches including Ecore as Eclipse’s variant, whereas on the Ontoware side the W3C description languages of the OWL family seem to be the most important.

In this paper, we use *UML and OCL* [7] as representatives for Modelware and *description logics (DL)* [8] as the respective representation for Ontoware². We shortly introduce both approaches, show that they can be mapped to by graph-based semantics, and discuss the three kinds of bridging based on this mapping.

The paper is structured as follows. Sections 1 and 2, respectively, give a short introduction to both technological spaces, explaining their main characteristics and sketching their formal bases. Section 3 discusses how both worlds can be mapped, and Section 4 explicates and discusses different ways of bridging them. Section 6 concludes the paper.

1. Modelware

Modeling has a long history in database and software engineering. A long chain of seminal work on modeling starting with Chen’s ERM [10], over several variants like NIAM [11], James Martin’s approach [12], the books of the Booch [13], Rumbaugh et al. [14], Jacobson [15] and many others lead to the current de-facto standardization given by

²Description logics are the pure logical formalisms behind ontology languages, like OWL 2 [9]. Since DL ontologies are much more compound than OWL ontologies, they are used here for brevity’s sake.

OMG's *Unified Modeling Language* (UML) [7], whose class and object diagrams offer means to notate conceptual models. Though UML [7] is quite helpful as a “unifying” approach, there also are other, richer technologies in Modelware (e.g. ADOxx [16], Ker-meta [17], and the TGraph Approach [18]) that have more expressive modeling elements.

Much work on software development (including model driven development [19] and model transformation [20]) and much implementation work (including several UML tools and the Eclipse³ project) build upon UML, mostly driven by the Object Management Group (OMG)⁴. Thus, a rich and well understood “technological space” has developed which is widely accepted and gets steadily extended by the Modelware community.

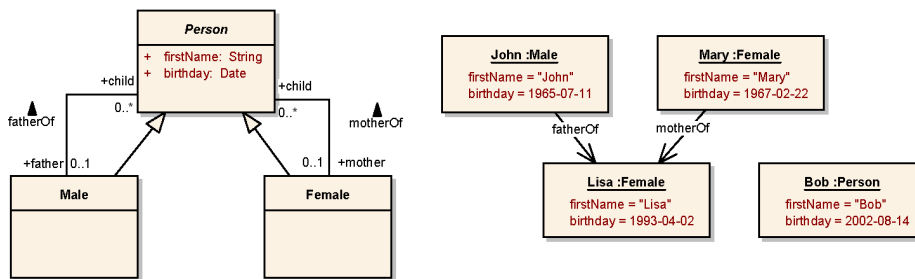


Figure 1. Two UML Diagrams

Example. To underpin the messages of this paper the often used and easy-to-understand family example is used. Figure 1 depicts a model of a genealogy as a *UML class diagram*. The description consists of classes (*Person*, *Male*, *Female*), which represent sets of objects, and associations (*fatherOf*, *motherOf*), which represent sets of links between these objects. Both, classes and associations, are classifiers, i.e. they have a name and stand for sets of instances. Classes may have named attributes with a respective domain, and they may generalize others (*Person* is a generalization of *Male* and *Female*). A corresponding set of instances is described by a *UML object diagram* in Figure 1. Here, the classes of the objects and the associations of the links are added as their types, and values are included for the type-specific attributes.

To define further restrictions for the sets of instances, additional constraints may be added. Class diagrams support the notation of *multiplicities* for associations which have been added beside the roles in Figure 1. Note that ‘0..1’ (instead of ‘1’) has been added to the parent roles, since a *closed world* is assumed by UML, which implies that all instance sets are finite and therefore there must be *Persons* whose *father* or *mother* is not in the instance set.

OMG proposes to use *OCL* [21] for the definition of *further constraints*. E.g., to describe that parents are born before their children, the following constraint might be added:

```
context Person
inv: father.birthday < self.birthday
```

³<http://www.eclipse.org>

⁴<http://www.omg.org>

```
and mother.birthday < self.birthday
```

Formal Foundation. Starting with Chen's [10] work, the formal semantics of models has mainly been defined using constructive set theory and predicate logics, leading to a clear *model-theoretic semantics*: For a given diagram D its extension, i.e. the set of its instances, is described.

Graphs are a means for talking about semantics which is widely used in Modelware. A concise description of semantics can be given using *graphs* as instances [22]. With the advent of MDA graph-based views on modeling semantics have become even more important, since graphs form an easily understandable, mathematically sound and efficiently implementable basis for storing and manipulating model instances. Many Modelware tools are based on graphs, especially when model transformations are concerned (e.g., AGG [23], PROGRES [24], MOLA [25], GReTL [26]).

A model-theoretic semantics for models defines the set of legal instances that are described by a model. In Figure 1 one instance graph of the genealogy model is sketched by the object diagram in the same figure.

Instance graphs have objects of the respective diagram classes as *vertices* and links corresponding to the respective diagram associations as *edges*. The edges are assumed to be directed according to their reading direction as expressed by the small black triangles at their association. The classes and associations serve as *types* of the instance vertices and edges. The types of the edges and their respective endpoints are compatible with the incidences between associations and classes in the model. Vertices may carry attribute-value-pairs as defined by their type (and all its supertypes to include inheritance) in the model.

Based on such a formal, unambiguous semantics the set of models defined by a conceptual diagram can be restricted by additional logical formulae. Thus, the formulae of the constraint language lead to additional constraints on the graph instances. Using the model-theoretic semantics of Schmitt [27], OCL constraints can directly be checked on instance graphs. To enable checking of the instance graph, *constraint languages* usually are algorithmically feasible sublanguages of predicate logic, which allow for model checking by assessing the conformance of instance sets. This also holds for OCL.

Summarizing, Modelware separates structural modeling from (additional) constraints by using different description paradigms, namely object-relationship descriptions and predicate logics which together define the legal sets of instances. Though the usage of constraint languages in practice does not seem to be so wide-spread as pure class diagrams, in principle the Modelware modeling facilities can be characterized as a combined use of visual class models and additional constraints which can be described precisely using a model-theoretic semantics.

2. Ontoware

The use of logics to describe conceptual models has a long history in works on knowledge-based systems in Artificial Intelligence. With the advent of the Semantic Web special logics like F-logic [28] and description logic (DL) [8] have been adopted. The former may be characterized as a strict format for descriptions in predicate logic, thus

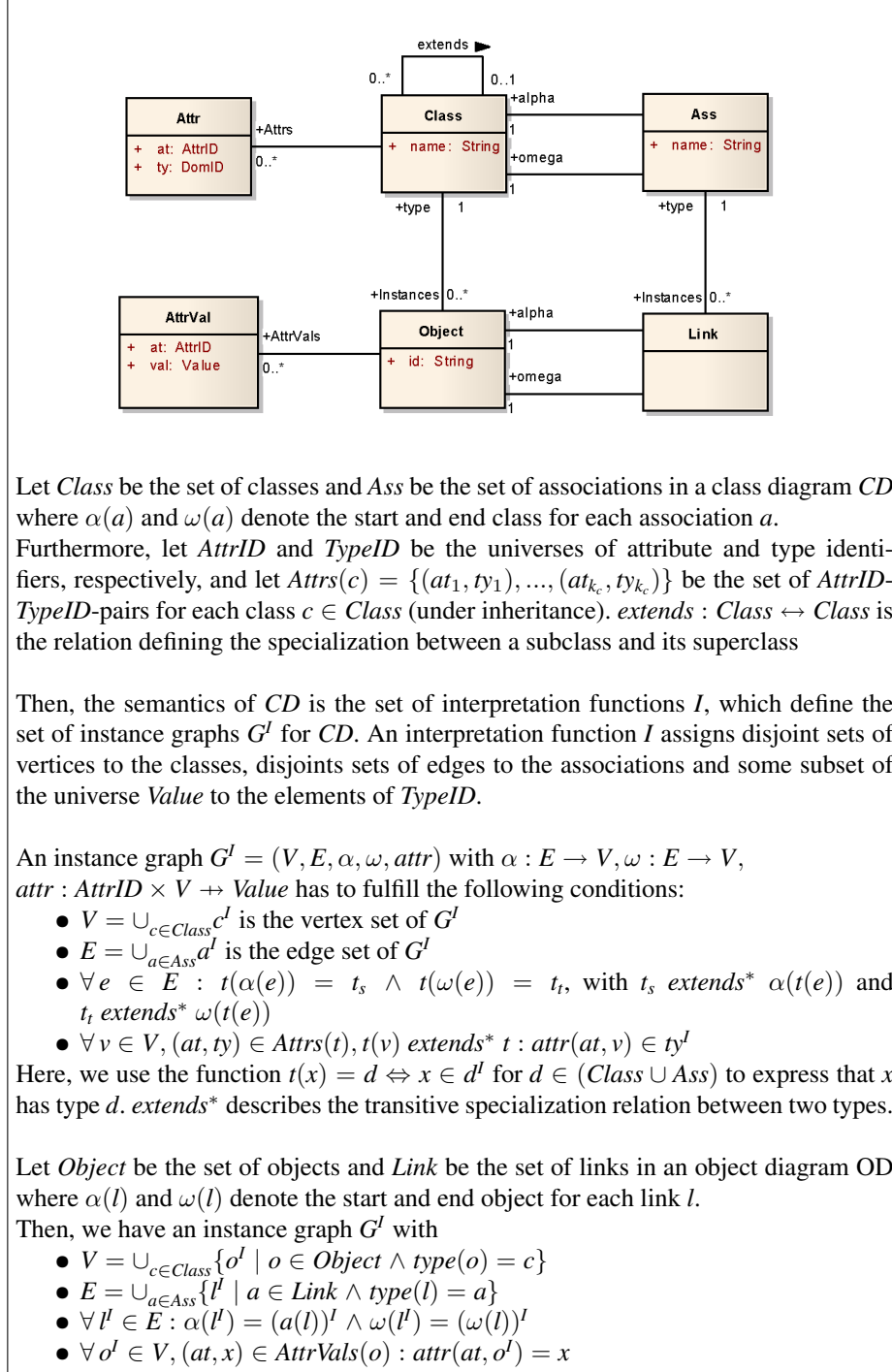


Figure 2. Graph Semantics for Models (Simplified Excerpt)

being undecidable in general. The latter is a family of sublanguages of predicate logic most of which are decidable having varying degrees of complexity.

The family of *description logics* [6] may be structured along a set of features described by calligraphic letters \mathcal{ALC} plus some of $\{\mathcal{F}, \mathcal{S}, \mathcal{H}, \mathcal{R}, \mathcal{O}, \mathcal{I}, \mathcal{N}, \mathcal{Q}\}$ which may be used to define a concrete DL language by the concept constructors it supports. Different sets of features lead to different expressivity and different complexity properties.

OWL (Web Ontology Language) is a family of precisely defined description logics introduced by the World Wide Web Consortium (W3C⁵). OWL documents are called *ontologies* by W3C. OWL comes with a set of corresponding notations and tools. The current version OWL 2 corresponds to the \mathcal{SROIQ} -subset of description logics. OWL 2 as a web ontology language is primarily defined by its abstract syntax [9]. Mostly, OWL 2 ontologies are serialized using RDF [29] as XML-documents. There are also some concrete syntaxes (e.g., Manchester style, functional style) which are better accessible to humans than pure XML. There are several sublanguages of OWL, called profiles, namely OWL 2 EL, OWL 2 QL and OWL 2 RL [30], which differ in their complexity. The OWL ontology languages are based on description logics (DL) [8].

Example. In description logics knowledge is described in two parts. The terminology component (*T-Box*) describes the concepts and their roles, whereas the assertion component (*A-Box*) specifies the individuals and their properties. Given the \mathcal{ALC} variant of DL, the model depicted in Figure 1 can be described as a DL-text in Figure 3 by axioms and assertions.

```
// T-Box
Male  $\sqsubseteq$  Person  $\sqcap \forall$  fatherOf . Person
Female  $\sqsubseteq$  Person  $\sqcap \forall$  motherOf . Person
Person  $\sqsubseteq \forall$  name . String  $\sqcap \forall$  birthday . Date

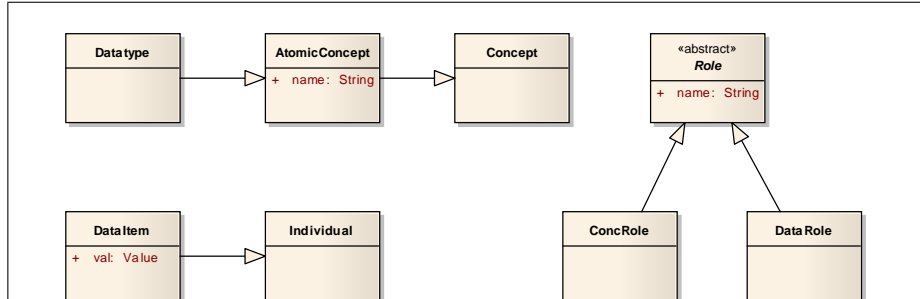
// A-Box
Person(John), Person(Mary), Person(Lisa), Person(Bob)
fatherOf(John, Lisa), motherOf(Mary, Lisa)
firstName(John, "John"), birthday(John, 1967-07-11)
firstName(Mary, "Mary"), birthday(Mary, 1967-02-22)
firstName(Lisa, "Lisa"), birthday(Lisa, 1993-04-02)
firstName(Bob, "Bob"), birthday(Bob, 2002-08-14)
```

Figure 3. A Description Logic Example

The first part of the description in Figure 3 is a T-Box which introduces the classes and their properties which are called *concepts* and *roles*, respectively, in pure DL. There are atomic concepts (Person, Male, and Female) and atomic properties (fatherOf, motherOf, name, and birthday). Further concepts are defined using the *constructors* $\forall r.c$ and $c \sqcap d$ for roles r and concepts c and d .

The *T-Box* consists of three *axioms*, each of which makes a statement about different (named or anonymous) concepts, e.g., it is stated that Males are Persons and all individuals they may be fatherOf are also Persons. Furthermore, it is stated that Persons

⁵<http://www.w3c.org>



Let *AtomicConcept* be the set of concepts and let *Role* be the set of roles in an *ALC* DL ontology *O*. Assuming *O* is schema-like, the roles are either concept roles (*ConcRole*) or datatype roles (*DataRole*).

Then, the semantics of *O* is the set of interpretation functions *I*, which define the set of instance Graphs G^I for *O*.

An instance graph $G^I = (V, E, \alpha, \omega, attr)$ with $\alpha : E \rightarrow V$ and $\omega : E \rightarrow V, attr : AttrID \times V \mapsto Value$ has to fulfill the following conditions:

- $\forall c \in AtomicConcept : c^I \subseteq V$
- $\forall r \in ConcRole : r^I \subseteq E$
- $\forall s \in DataRole : s^I : \{s\} \times V \mapsto Value$

Furthermore, there are additional conditions for the concept constructors in *ALC* $\forall c, d \in Concept, r \in ConcRole, s \in DataRole$:

- $\top^I = V$
- $\perp^I = \emptyset$
- $(c \sqcap d)^I = c^I \cap d^I$
- $(c \sqcup d)^I = c^I \cup d^I$
- $(\neg c)^I = V \setminus c^I$
- $(\forall r.c)^I = \{v \in V \mid \forall w \in V, e \in r^I : v = \alpha(e) \wedge w = \omega(e) \Rightarrow w \in c^I\}$
- $(\exists r.c)^I = \{v \in V \mid \exists e \in r^I : \alpha(e) = v \wedge \omega(e) \in c^I\}$
- $(\exists s.c)^I = \{v \in V \mid \exists x \in Value : attr(s, v) = x\}$

According to the *axioms* in the ontology, the instance graph has to fulfill further conditions $\forall c, d \in Class$:

- $c \sqsubseteq d \Rightarrow c^I \subseteq d^I$
- $c \equiv d \Rightarrow c^I = d^I$

Let *Individual* be the set of individuals in *O*. Then we have $Individual^I \subseteq V$.

Finally, the *assertions* imply some constraints:

- $\forall (c, i) \in ConceptAssertion : i^I \in V$
- $\forall (r, i, j) \in RoleAssertion : \exists e \in r^I : \alpha(e) = i \wedge \omega(e) = j$

Figure 4. Graph Semantics for DL (*ALC*, Simplified Excerpt)

may have a `firstName`, which is a `String`, and a `birthday`, which is a `Date`. Note, that relationships like `fatherOf` and attributes like `name` and `birthday` are not differentiated. They both are treated equally by roles. The *A-Box* enumerates the individuals and their properties using concept and role assertions.

Formal Foundation. The meaning of ontologies is precisely defined in a comprehensive logic theory. Looking at them as models in the sense described above, a *model-theoretic semantics* seems to be the most adequate description for the purpose of bridging Ontoware and Modelware.

The model-theoretic semantics can also be given by *graphs*. Concepts (`Person`, `Male`, `Female`) correspond to vertex sets (which do not have to be disjoint), and roles stand for directed edges. Concepts are *not* viewed as types, but only as sets of vertices. Thus, a vertex can belong to several concepts. Consequently, subsumption (\sqsubseteq) corresponds to set inclusion and not to subtyping.

Figure 4 sketches a graph semantics, which is defined inductively over the set of concept constructors. Note, that in this case, the semantics describes only the inclusion of the interpretation sets in the graph. Since an *open world semantics* is assumed, there may be more vertices and edges in an instance graph than the enumerated individuals. In fact, the graph may even be infinite.

Roles correspond to attributes of the vertices, if their object is a basic value, and to edges otherwise. Assuming that `String` and `Date` are basic values, all graphs conforming to Figure 1 do also conform to Figure 3. The contrary is not necessarily true, since the open world assumption allows further vertices and edges in the instance graphs.

Whereas Modelware is based on explicit classes and explicit expression of *specialization*, Ontoware uses concepts as a general means for expressing properties. Thus, there are additional anonymous concepts and specialization (in the form of *subsumption*) is an implicit property.

3. Mapping

Conceptual models describe a domain in terms of its concepts and their relations and possibly also by sets of suitable instances and links. Both, UML class diagrams with constraints and object diagrams as well as description logic-based languages like OWL 2 offer means to describe conceptual models formally. Though both approaches stem from different technological space they have a lot of *commonalities*, as has been shown in Sections 1 and 2. Thus, there is some evidence, that bridging both worlds might be possible, i.e. that models and ontologies may be mapped to each other.

Any kind of bridging has to be based on a *mapping* of the notions of both worlds. The model-theoretic semantics sketched in Sections 1 and 2 show that the relevant parts of both worlds match quite well and give hints how such a mapping may be done. Assuming that the ontologies used as conceptual models in software development are *schema-like*, the graph-based semantics defined for Modelware and Ontoware, respectively, can be used as a basis for mapping. An approximate matching of notions derived from the semantics is visualized in Figure 5.

The graph-based semantic description in Sections 1 and 2 render a high similarity between both (simplified) modeling approaches, namely UML class diagrams, OCL and object diagrams on the one hand and description logical models (as representatives of on-

UML	DL (schema-like)
Class	Atomic Concept
Association	Concept Role
Attribute	Datatype Role
Object	Individual
Domain	Data Type
Subclassing	Subsumption

Figure 5. Equivalences between Models and Ontologies

tologies) on the other hand. This similarity paves the way to defining a mapping between both worlds as a basis for a bridge.

- Classes in UML correspond to concepts in DL. They both stand for a specific set of vertices in the instance graph.
- The associations in UML correspond to concept roles in DL. They both stand for specific sets of edges in the instance graph.
- The attributes in UML correspond to datatype role in DL, since they associate objects to values from a predefined universe.
- The UML objects represent single vertices like the individuals in DL descriptions.
- The domains of attributes in UML comprise atomic values like the datatypes in DL.
- Subclassing in UML implies subsumption of the respective instance sets, but it furthermore implies inheritance, so this correspondence is not fully symmetric.

4. Bridging

Given a mapping like the one described in Section 3, a common understanding of the two different technological spaces under consideration is given and a formalization of the commonalities and the variabilities of both worlds becomes feasible.

Bridging, i.e. combining different technological spaces, can be achieved in many different ways, depending on the context given for software development. But, bridging does not only pertain to the respective model and ontology *artifacts*, including their logical background and their languages. It has also to pertain to all *services and tools* belonging to the respective world, e.g., query and transformation languages.

Assuming, for instance, software development is done in some Modelware environment (e.g., in an Eclipse/Ecore/EMF world), it might make sense to use some specific Ontoware ontology for reusing some widely spread ontology for some special purpose (e.g., some domain ontology provided by the community). In this case, the use of that ontology by some API might be an appropriate approach. This approach uses only a loose coupling to the Ontoware world, from which only some well-defined services are used. We call this kind of bridging *adaptation*, since the API constitutes some kind of adapter that makes the ontology fit into the modeling world. But still both worlds coexist. Adaptation may also be used in the opposite direction.

If for some reason a pure homogeneous world is aspired, coexistence has to be replaced by some kind of *transformation* that generates a new model in the target space from a given model in the source space, e.g., a UML diagram may be used as a light-weight ontology in some Ontoware environment by generating a corresponding OWL

ontology from it. Note, that this kind of bridging usually implies some loss of information, since both worlds have different properties not all of which are transformable into the other world.

The third way of tackling the bridging of two different spaces is the *integration* of both spaces into one all-embracing new technological space. To achieve this, the mapping of concepts (which defines some kind of *intersection* of both worlds) has to be used to define a new target technological space which corresponds to the *union* of the source spaces. Apparently, such an integration affords a deep understanding (and thus a lot of research) and a good set of design decisions in the case of conflicting properties of both spaces.

In this section, we give short characterization of these three kinds of bridging approaches in terms of service definitions that specify the target result on an abstract level.

4.1. Adaptation

The first approach to bridging is keeping both technological spaces as they are, but make their *services* available in a common form. Given a mapping between the respective notions, a common service interface can be defined that acts as an adapter and allows an immediate comparison between both worlds.

The usefulness of a conceptual model depends on the services that come along with it. For application software that makes use of conceptual models, a *well-defined API* may grant use of such services in a generic way.

Figure 6 gives an overview on typical services supplied by conceptual models. The *specifications* given there are 'high-level' in the sense that they try to describe the services on a level of abstraction that (i) is precise enough to be understandable by experts and (ii) does not dive too much into the notational depth of a programming language. The specification assumes a class `ConceptualModel` which is an abstract superclass of classes `Model` and `Ontology` which implement the respective services.

Modelware offers models as versatile data structures that can be manipulated, analyzed, traversed, queried, and transformed into other models, whereas the focus in *Ontoware* is on the use of reasoning services on the model that come along with description logics and make entailed knowledge available.

General Services. There are numerous tools that support *editing* of conceptual models in both worlds. Though there are more UML editors around than OWL editors, the integration of OCL-like constraints into UML editors is still not widely supported yet, whereas OWL includes the axiomatic parts because of the homogeneous character of description logics.

Query Answering is handled differently in both worlds. The type Query applies to some artifact that contains a query in some well-defined query language (e.g. OCL, GReQL, or SPARQL). *Models* can be queried by efficient query languages (depending on their internal representation) that are in the tradition of database technology. This holds for OCL [21] and especially GReQL [31] which is a graph query language. *Ontologies* are usually queried after a closed world completion by a reasoner which is an additional step that enlarges the ontology by making some entailed information manifest in the ontology. Thus, additional preprocessing is necessary. SPARQL [32] is a language for querying ontologies.

General Services.

Name	<i>Editing</i>
Signature	void edit ();
Description	supports external editing of the conceptual model <i>cm</i>
Name	<i>Query Answering</i>
Signature	String query (Query q);
Description	returns a string <i>s</i> which contains the result of applying the query <i>q</i> to the conceptual model <i>cm</i> (according to the semantics of the query language)
Name	<i>Consistency</i>
Signature	boolean checkConsistency ();
Description	checks whether the conceptual model <i>cm</i> is consistent
Explanation	a conceptual model <i>cm</i> is <i>consistent</i> iff there exists a non-empty instance graph for <i>cm</i> .
Name	<i>Instance Consistency</i>
Signature	boolean checkInstanceConsistency (Instance i);
Description	checks whether an instance graph <i>i</i> conforms to the conceptual model <i>cm</i>

Reasoning Services.

Name	<i>Satisfiability</i>
Signature	boolean checkSatisfiability (Class c);
Description	checks whether a class <i>c</i> is satisfiable in the conceptual model <i>cm</i>
Explanation	a class <i>c</i> is <i>satisfiable</i> (instantiable) in conceptual model <i>cm</i> iff there is an instance graph for <i>cm</i> with a non-empty set of vertices belonging to <i>c</i> . (Note, that satisfiability is a property of specific class. A conceptual model may contain non-satisfiable classes, while still being consistent.)
Name	<i>Subsumption</i>
Signature	boolean subsumes (Class c1, Class c2);
Description	checks whether a class <i>c1</i> subsumes a class <i>c2</i> in a conceptual model <i>cm</i>
Explanation	a class <i>c1</i> <i>subsumes</i> a class <i>c2</i> iff all instance vertices of <i>c2</i> are also instances of <i>c1</i> in all models.
Name	<i>LeastCommonSubsumer</i>
Signature	Class getLeastCommonSubsumer (Set<Individual> s);
Description	returns the smallest (named) class <i>c</i> that all individuals in <i>s</i> are instances of
Name	<i>Classification</i>
Signature	boolean classifies (Class c, Individual i);
Description	checks whether an individual <i>i</i> is an instance of a class <i>c</i> in a conceptual model <i>cm</i>

Figure 6. Services Offered as Methods by Conceptual Models

Consistency in general checks whether there is a non-empty instance graph for a model. This is a well-known reasoning service for ontologies, which is usually not offered for models. In contrast to that, *Instance consistency* (also called A-Box consistency) only checks an instance graph for its conformance to the model. This service is offered in both worlds, since it can be interpreted as a boolean query. In Ontoware it is usually done in an open world manner.

Conceptual Model transformation.

Name	<i>Transformation Model to Ontology</i>
Signature	Ontology transform(Model m)
Description	transforms a model to a schema-like ontology
Name	<i>Transformation Ontology to Model</i>
Signature	Model transform(Ontology o)
Description	transforms the structural part of an ontology <i>o</i> to a model

Query transformation.

Name	<i>Transformation Model Query to Ontology Query</i>
Signature	OntologyQuery transform(ModelQuery q)
Description	transforms a model query <i>q</i> to an ontology query
Name	<i>Transformation Ontology Query to Model Query</i>
Signature	ModelQuery transform(OntologyQuery q)
Description	transforms an ontology query <i>q</i> to a model query

Figure 7. Transformation Service between Modelware and Ontoware

Reasoning Services. There are some services that rely on reasoning techniques and are usually not offered by models. Reasoning goes beyond querying in the sense that implicit knowledge is inferred, whereas classical querying can refer to explicit knowledge only. *Satisfiability*, *subsumption*, and *classification* are typical ontology services that are usually not available on models.

Further Services. Besides the described services offered by conceptual models, there are more and more manipulation services offered today that act on them. With the advent of MDA, model *transformation* has become an important means for modeling. Also model *merging*, model *differencing*, and model *patching* are important for the support of model versioning. Though this work can be done for ontologies, as well, there seems to be not so much activity there in that respect, yet.

4.2. Transformation

Based on a mapping of elements of models and ontologies (Figure 5), it is quite easy to read UML-diagrams as schema-like ontologies. On the other hand, schema-like ontologies can also be translated to UML-diagrams as far as their structural part is concerned.

The transformation services specified in Figure 7 transform a **Model** according to a transformation definition which is based on a mapping like the one in Figure 5 to an **Ontology**. It creates for each class in a model a concept in the ontology. All associations and attributes are transformed to corresponding roles. Because UML class diagrams and ontologies contain instances of classes and concepts, respectively, objects of a model are transformed to individuals of an ontology. The transformation builds for each domain definition in a model a datatype in the ontology. For the subclassing relations in models, the transformation builds a subclass axiom in the ontology. The reverse transformation service realizes the translation from ontologies to models.

The translation of *axioms* in ontologies by the transformation service from ontologies to models in Figure 7 has to be treated more carefully. OCL-like axioms may use *set-theoretic constructs* based on an algorithmically feasible subset of predicate logic that may not be supported by the respective variant of the DL. On the other hand, class

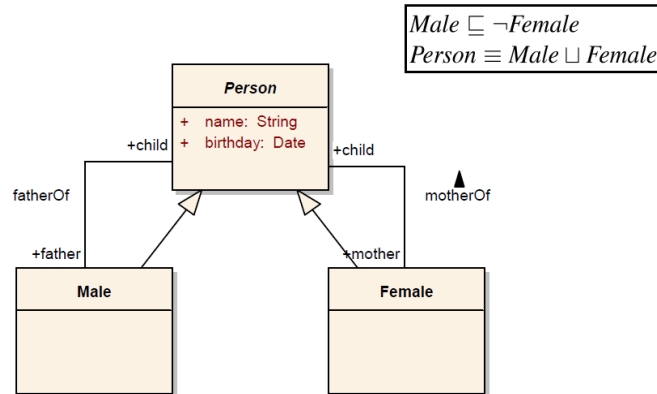


Figure 8. Hybrid Person Model

expressions in OWL (which define anonymous classes) may lead to *anonymous classes* in the UML world which might appear quite unnatural from the model point of view.

[33] describes a simple and a schema-aware translation from schema-like RDF graphs (which may be used for representing ontologies) to a graph-based variant of UML and backward.

Beside transformations of conceptual models a transformation bridge between model and ontology query languages can be established. The abstract specification of such transformation services is shown in Figure 7. [33] describes the mutual transformation of the respective query languages GReQL for models and SPARQL for ontologies.

Having a transformation service established, language users are able to build models with the modeling language (e.g. UML) and concrete syntaxes they are familiar with. The users are restricted to the expressiveness (e.g. different syntactic constructs) of the modeling language they are working with. The transformation services between query- and constraint languages increase the expressiveness of models to be transformed by the bridge, since they consider the models as well as annotated constraints or queries.

The tools and modeling environments for creating and processing models and ontologies remain the same. The transformation bridge establishes interoperability between modeling environments and different ontology tools (like reasoners).

4.3. Integration

The strongest form of bridging is the integration of both technological spaces into one single integrated one. Based on the mapping in Figure 5, it is possible in principle to integrate the *metamodels* of both worlds (assuming they are written in a common metameta-language like MOF) by identifying (or subclassing) the respective notions. Then, the additional notions from both worlds may be added leading to a hybrid modeling language which could be able to use the best of both worlds.

A hybrid model conforms to a hybrid modeling language and is built by constructs of both languages. In the example in Figure 8 the UML class diagram from Figure 1 is extended by some DL axioms. It expresses, that the two concepts **Male** and **Female** are disjoint and the set of instances described by the concept **Person** is equivalent with the union of the two sets of instances described by **Male** and **Female**.

Figure 9 presents the basic services for language designers, building hybrid modeling languages, and for language users, using the hybrid modeling language.

To establish a hybrid modeling language the metamodels of both languages are combined. Figure 9 depicts an abstract specification of an integration services. It gets as input two metamodels and returns an integrated metamodel.

A concrete implementation of the integration service for the UML metamodel and the OWL metamodel affords a lot of design decisions, since comparability of notions does not immediately lead to a smooth integration. Language designers have to manually implement this metamodel integration service which results the integrated UML+OWL metamodel. Therefore they rely on a set of basic integration operations [34]. These operations allow for *merging* two classes, if the classes have the same meaning and are equivalent conceptually. They allow for *subclassing* two classes, if one class describes a more general concept than the other, or they just *establish an association* between two classes if the concepts are just related. Its use is based on intensional knowledge of the languages to be combined, and a mapping relating similar constructs of both worlds.

For the metamodels given in Figure 2 and 4, we suggest to merge UML class and DL concept, because in both technological spaces they have similar semantics, since both describe sets of instances. In addition, the UML association construct is combined with the DL concept role and the UML attribute is combined with the DL data role. Associations and concept roles describe sets of relations between instances. The attribute and the data role describe sets of relations between instances and data values.

An integrated metamodel is being developed in MOST. It is based on the TwoUse approach [35] and supplies a basis for conceptual modeling that combines the facilities of both worlds.

For language designers and users the interoperability with other tools is important. In particular, language users having created a hybrid model want to project it to a pure ontology which serves as input for several reasoning tools.

We propose the implementation of projection services. Figure 9 depicts the specifications of projection services. Both get as input a hybrid model (conforming to a metamodel designed by the integration service).

The ontology projection service projects the hybrid model to an ontology. Since we have integrated the UML class with the DL concept, the ontology projection service projects all classes in Figure 8 to pure DL concepts in the new ontology. In addition all associations and attributes depicted in Figure 8 are projected to DL concept roles and data roles. The additional concept constructors (for union and complement) conform to ontology constructs and are projected without any change to the new ontology.

The model projection service returns a new model which is built by the class diagram in Figure 8.

From the perspective of language users, the integration bridge provides a common view together on modelware and ontoware conceptual models, namely hybrid models. The modeling of hybrid models requires an understanding of both integrated languages. A language user has to be familiar with different concrete syntaxes (at least one for each modeling language) and how they are used in combination. To provide interoperability between different tools, projection services must be given by the integration bridge. Projection services extract all relevant information from hybrid models and translate them to models understandable by given tools. For example hybrid models are projected to ontologies to be readable by reasoners.

Abstract Integration Service.

Name	<i>Integration of Metamodels</i>
Signature	Metamodel integrate (Metamodel m_u , Metamodel m_o)
Description	integrates two metamodels m_u and m_o . The result is an integrated meta-model.

Integration Operations.

Name	<i>Merge Integration Service</i>
Signature	Class merge (Class c_1 , Class c_2)
Description	merges the two classes c_1 and c_2 by replacing them by a new class. All incidences with associations and specialization relations and all nested attributes are moved to the new class.

Name	<i>Specialize Integration Service</i>
Signature	void specialize (Class c_1 , Class c_2)
Description	creates a specialization relationship between two classes c_1 and c_2 .

Name	<i>Associate Integration Service</i>
Signature	Association associate (Class c_1 , Class c_2)
Description	associates two classes c_1 and c_2 by a newly created association

Projection Services.

Name	<i>Ontology Projection Service</i>
Signature	Ontology project _o (ConceptualModel m_c)
Description	creates a new ontology m_o which only consists of those parts of m_c which conform to ontology constructs defined in the metamodel of an ontology language (e.g the one in [9]).

Name	<i>Model Projection Service</i>
Signature	Model project _m (ConceptualModel m_c)
Description	creates a new model m_m which only consists of those parts of m_c which conform to UML class diagram constructs defined in the class diagram meta-model.

Figure 9. Integration and Projection Services

5. Discussion

A comparison between Modelware and Ontoware is always subjective to some extent since the results have been developed in different scientific communities with different goals and different terminologies. In essence, the support used by a bridging approach has to be chosen carefully depending on the features needed.

5.1. Comparing Modelware and Ontoware

As the descriptions above may have shown comparison of Modelware and Ontoware is a multi-faceted issue which has several dimensions: (a) logics, (b) languages, (c) tools, and (d) conceptual modeling styles. Thus, depending on the situation and on the requirements of the actual application different combinations of logics, languages, tools, and styles might be appropriate.

(a) logics: Modelware and Ontoware both provide declarative means to describe conceptual models. While Modelware is based on *constructive set theory*, whose descrip-

tions consist of a structural part and additional constraints, Ontoware is based on *description logic*, which allows the joint description of both parts. The subtle difference entailed by the open world assumption in ontologies (as opposed to the closed world assumption in models) leads to different results e.g. for consistency checking, which may be quite useful but has to be used with care.

(b) languages: To come to concrete statements about similarities and dissimilarities between both worlds, concrete languages have to be chosen for describing structure and constraints for models on the one hand and for the ontologies on the other hand. As DL theory suggests, several variants of descriptions logics may be used with differing properties. The OWL 2 family of ontology languages offers several such profiles.

(c) tools: Both worlds are supported by tools. There are *visual editors* for models (various UML tools) and *textual editors* for ontologies (e.g. protégé). Since models also act as prescriptions, *code generation tools* are available, as well. There seem to be much more manipulation tools around on the Modelware side especially stimulated by the needs of MDA. This also pertains to the respective query and manipulation languages (for transformation, differencing, patching, etc.). The prescriptive aspect of ontologies primarily pertains to their *reasoning facilities*, when they are used via their runtime services. While classical modeling is more efficient for querying and A-Box constraint checking, ontologies supply simultaneous T- and A-Box checking and open world reasoning.

(d) styles: The concrete editing, querying and reasoning *technologies* applied surely imply different modeling styles and the use of different services. In general, Modelware languages have a stronger typing schema and come with more constraints on the model structure than than general predicates. Therefore more disciplines and conventions are necessary to keep ontologies well-structured. Especially for ontologies, the *modeling style* used influences the comparison. As stated above, schema-like modeling eases the use of class diagrams as a lightweight description of the concept structure and allows a schema-aware transformation of query languages.

On the other hand, there is a well-developed body of knowledge for modeling styles, including collections of *design patterns*, catalogs of *best practice*, and work on the evaluation of *quality of models*, which is not directly usable for the corresponding ontologies.

5.2. Comparing Bridging Approaches

To bring both worlds together, this paper distinguishes three different bridging approaches, namely (a) adaptation, (b) transformation, and (c) integration. Each of these approaches has different advantages and disadvantages.

(a) adaptation: The definition of application programming interfaces (APIs) which make the common usage of Modelware and Ontoware facilities possible is the current state of the practice. All reasoners supply such an API for the construction of ontologies and the respective reasoning services. But still the interoperability between tools for constructing and editing models or ontologies, respectively, and application software that makes use of these models at runtime, is far from being standardized.

(b) transformation: Tools for transforming conceptual models and their corresponding constraints and queries from one world into the other are quite helpful for many applications. But the fact that both worlds have properties that are not directly present in the other world shows the limitations of this approach.

(c) integration: The probably best way to bridge both worlds seems to be an amalgamation of Modelware and Ontoware. The common kernel of both worlds sketched in this paper shows that it should be possible to define support for conceptual modeling which has the strength of schema-like modeling including full constraint checking (from Modelware) as well as the facilities of reasoning (from Ontoware). Though the fact that both worlds are well-established with their own communities, languages, techniques, and tools makes such a development hard, following the presentation from above such an integration still seems to be accomplishable.

6. Conclusion

This paper gave some impressions on the comparison of two different technological spaces for conceptual modeling, namely the UML/MOF world, called *Modelware* for short, and the OWL world, called *Ontoware*. It turned out that the comparison of Modelware and Ontoware is a multi-faceted issue, since there are many alternative features to choose from on both sides.

Though bridging seems reasonable in principle, there are still subtle differences in the use of both technological spaces which also were shortly discussed. It turned out that a definite decision is only possible if a multidimensional comparison is done on concrete implementations.

The work presented in this paper shows that in principle it is possible to create a common technological space supplying the united power of Modelware and Ontoware.

Simultaneously, looking at the diversity of languages, techniques and tools and the quite small amount of research done in uniting both worlds as well as the quite disjoint communities and the different consortia (OMG and W3C) there is still a long way to go.

Acknowledgement

The author would like to thank the MOST team for the productive working atmosphere in this project. Most of the statements in this paper have their roots in discussions with the consortium. Special thanks go to the Koblenz team, especially to Hannes Schwarz, Steffen Staab, Fernando Parreiras and Gerd Gröner.

References

- [1] Kurtev, I., Bézivin, J., Aksit, M.: Technological spaces: An initial appraisal. In: International Symposium on Distributed Objects and Applications, DOA 2002. (2002)
- [2] Ebert, J.: Software engineering with models and ontologies. In Barzdins, J., Kirikova, M., eds.: Databases and Information Systems, Riga, University of Latvia Press (2010)
- [3] Walter, T., Parreiras, F.S., Staab, S., Ebert, J.: Joint Language and Domain Engineering. In: Proc. of 6th European Conference on Modelling Foundations and Applications, ECMFA 2010, Paris. Volume 6138 of LNCS., Springer (2010)
- [4] Walter, T., Silva Parreiras, F., Staab, S.: OntoDSL: An Ontology-Based Framework for Domain-Specific Languages. In: ACM/IEEE 12th International Conference on Model Driven Engineering Languages and Systems, 12th International Conference, MODELS 2009. Volume 5795., Springer (2009) 408–422
- [5] Gruber, T.R.: A translation approach to portable ontology specifications. Knowledge Acquisition 6 (1993) 199–221

- [6] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Pater-Schneider, P., eds.: *The Description Logic Handbook*. Cambridge University Press, Cambridge (2003)
- [7] OMG: *UML 2.0 Superstructure Specification (formal/05-07-04)*. Technical report (2005)
- [8] Baader, F., Nutt, W.: *Basic description logics*. In: *The Description Logic Handbook*. Cambridge University Press (2003)
- [9] W3C, ed.: *OWL 2 Web Ontology Language - Structural Specification and Functional-Style Syntax*. (2009)
- [10] Chen, P.P.S.: *The entity-relationship model—toward a unified view of data*. *ACM Trans. Database Syst.* **1**(1) (1976) 9–36
- [11] Verheijen, G.M.A., van Bekkum, J.: *Niam: An information analysis method*. In Olle, T.W., Sol, H.G., Verrijn-Stuart, A.A., eds.: *Information System Methodologies*, Amsterdam, North-Holland (1982)
- [12] Martin, J., McClure, C.: *Diagramming Techniques for Analysts and Programmers*. Prentice-Hall, Englewood Cliffs (1985)
- [13] Booch, G.: *Object-Oriented Design*. Benjamin/Cummings, Redwood City (1991)
- [14] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.: *Object-Oriented Modeling and Design*. Prentice Hall, Englewood Cliffs NJ (1991) Original.
- [15] Jacobson, I.: *Object-Oriented Software Engineering*. Addison-Wesley, Wokingham (1992)
- [16] Junginger, S., Kühn, H., Strobl, R., Karagiannis, D.: *Ein Geschäftsprozessmanagement-Werkzeug der nächsten Generation - ADONIS: Konzeption und Anwendungen*. *Wirtschaftsinformatik* **42** (2000) 392–401
- [17] Muller, P.A., Fleurey, F., Jézéquel, J.M.: *Weaving executability into object-oriented meta-languages*. In: *MODELS 2005*. (2005)
- [18] Ebert, J., Riediger, V., Winter, A.: *Graph Technology in Reverse Engineering, the TGraph Approach*. In Gimnich, R., Kaiser, U., Quante, J., Winter, A., eds.: *10th Workshop Software Reengineering (WSR 2008)*. Volume 126 of *GI Lecture Notes in Informatics.*, Bonn, GI (2008) 67–81
- [19] OMG: *MDA guide version 1.0.1*. Technical report (2003)
- [20] OMG: *MOF Query / Views / Transformations*. Technical report (2008)
- [21] OMG: *Object Constraint Language, v2.0*. Technical report (2006)
- [22] Walter, T., Ebert, J.: *Foundations of graph-based modeling languages*. Technical report, Dept. of Computer Science, University Koblenz-Landau (to appear, 2010)
- [23] Taentzer, G.: *AGG: A graph transformation environment for modeling and validation of software*. In Paltz, L., Nagl, M., Böhlen, B., eds.: *Applications of Graph Transformations with Industrial Relevance: Second International Workshop (AGTIVE 2003)*, Springer (2004) 446–453
- [24] Ranger, U., Weinell, E.: *The graph rewriting language and environment progres*. In: *Applications of Graph Transformations with Industrial Relevance*. vol. 5088 of *LNCS*, Springer (2008)
- [25] Kalnins, A., Barzdins, J., E.Celms: *Model transformation language MOL.A*. In: *Proceedings of Model-Driven Architecture: Foundations and Applications (MDAFA 2004)*. (2004) 14–28
- [26] Horn, T., Ebert, J.: *The GReTL transformation language*. Technical report, Department of Computer Science, University Koblenz-Landau, Koblenz (2010)
- [27] Schmitt, P.H.: *A model theoretic semantics of OCL*. (2001)
- [28] Kifer, M., Lausen, G.: *F-logic: a higher-order language for reasoning about objects, inheritance, and scheme*. *SIGMOD Rec.* **18**(2) (1989) 134–146
- [29] W3C, ed.: *Resource Description Framework (RDF): Concepts and Abstract Syntax*. (2004)
- [30] W3C, ed.: *OWL 2 Web Ontology Language Profiles*. (2009)
- [31] Ebert, J., Bildhauer, D.: *Reverse engineering using graph queries*. In Schürr, A., Lewerentz, C., Engels, G., Schäfer, W., Westfechtel, B., eds.: *Graph Transformations and Model Driven Engineering*. *LNCS* 5765. Springer (2010) to appear.
- [32] W3C, ed.: *SPARQL Query Language for RDF, W3C Recommendation*. (2008) <http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/>.
- [33] Schwarz, H., Ebert, J.: *Bridging Query Languages in Semantic and Graph Technologies*. In: *Reasoning Web. Semantic Technologies for Software Engineering*. Volume 6325 of *LNCS.*, Springer (2010)
- [34] Walter, T., Ebert, J.: *Combining DSLs and Ontologies using Metamodel Integration*. In: *Domain-Specific Languages*. Volume 5658 of *LNCS.*, Springer (2009) 148–169
- [35] Silva Parreiras, F., Staab, S.: *Using Ontologies with UML Class-based Modeling: The TwoUse Approach*. *Data and Knowledge Engineering* ([accepted for Publication], 2010)