On Marrying Ontological and Metamodeling Technical Spaces [*]

Fernando Silva Parreiras[0]
ISWeb — Information
Systems and Semantic Web,
Institute for Computer
Science,
University of Koblenz-Landau
http://isweb.uni-koblenz.de
parreiras@uni-koblenz.de

Steffen Staab[0]
ISWeb — Information
Systems and Semantic Web,
Institute for Computer
Science,
University of Koblenz-Landau
http://isweb.uni-koblenz.de
staab@uni-koblenz.de

Andreas Winter
Institute for Computer
Science, Johannes-
Gutenberg-University
Mainz
Staudingerweg 9
Mainz 55128, Germany
winter@uni-mainz.de

ABSTRACT

In software engineering, the use of models and metamodeling approaches (e.g., MDA with MOF/UML) for purposes such as software design or software validation is an established practice. Ontologies constitute domain models formalized using expressive logic languages for class definitions and rules. Hence, when seen from an abstract point of view, the two paradigms and their various technological spaces seem closely related. However, in the state-of-the-art research and practice the two technologies are just beginning to converge and the relationship between the two is still under exploration. In this paper, we give an outline of current ontology technologies, such as the Semantic Web standards for a Web Ontology Language (OWL). Then, we describe a domain analysis of the different technical spaces, explaining the features of the different paradigms. Eventually, we describe some avenues for integrating various ontological technical spaces into meta modeling technical spaces.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Computer-aided software engineering (CASE)*

General Terms

Languages, Design

Keywords

Model Driven Engineering, Meta Modeling, Ontology

1. INTRODUCTION

*This paper corrects one erratum that appeared in the version published with ACM.

In software engineering, model-driven techniques (Model-Driven Engineering, MDE) have gained broad acceptance over the last decade. Model-driven techniques provide management of, transformations between and synchronization of different models, including the "model" at the target platform constituted by source code. MDE is motivated by the objective of factorizing complexity into different levels of abstraction and concern, from high-level conceptual models down to specific aspects of the target platform.

An instance of MDE is the Model Driven Architecture (MDA) [31], which is based on OMG's Meta-Object Facility. It frequently includes UML as its modeling language and a common pipeline of managing and transforming models according to MDA [24] is depicted in the dashed box of Fig. 1: a platform-independent model (PIM) is transformed into a platform-specific model (PSM) and eventually into an executable representation (code) being the target platform. Thereby each transformation, i.e., each arrow, also indicates the enrichment of the resulting model with new features possibly specified in additional models.

Also over the last decade, the Web, AI and database communities have successfully investigated and promoted the use of *ontologies* as modeling and reasoning frameworks for the management of models and corresponding (Web) data. Ontologies and MDA technologies exhibit different foci. OMG MOF targets automating the management and interchange of metadata whereas knowledge representation focuses on semantics of the content and on automated reasoning over that content [20].

While the focus of these communities is somewhat different, the question still arises how the scientific and technical results around ontologies, ontology languages and their corresponding reasoning technologies can be used fruitfully in model-driven software engineering.

Like models, ontologies may provide a foundation for MDE. Thus, MDE can be based on the *Metamodeling Technical Space* (MMTS) as well as on *Ontological Technical Spaces* (OTSs). Kurtev et al. [28] have coined the term *technical spaces* to organize concepts in order to compare complex solution approaches. A technical space (TS) can be here understood as a body of knowledge comprising modeling languages and transformation facilities.
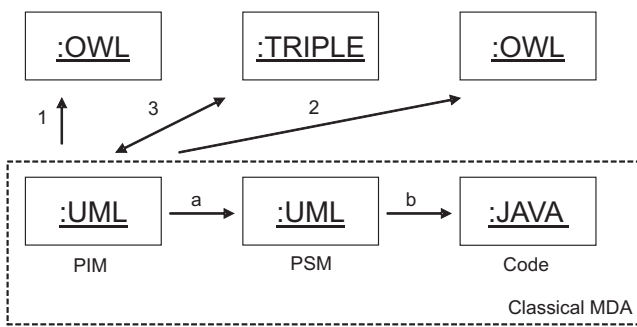
Figure 1: Marriage of MMTS and OTSs.

Subsequently, we will investigate the properties of ontological technical spaces, elucidating the potential of ontology technologies in MDE. Figure 1 illustrates an example indicating the use of several OTSs in the MDE process. The classical MDA transformations residing in the metamodeling technical space, such as explained above, are extended by further transformations making use of OTSs.

Further transformation into OTSs may provide additional analysis and implementation support, not as efficiently available in metamodeling technical spaces. Currently, MDA uses semi-formal metamodels instead of formal specification languages as support to describe models [48]. In Fig. 1, the initial UML model representing the PIM is transferred into an ontological representation in OWL, e.g., for model checking (arrow 1). This OWL model describes a submodel of the UML model, enabling logics-based model analysis. An second transformation translates the PIM into another OWL model (arrow 2). This model from OTS may serve as a kind of data base for a reasoner, invoked by the Java program.

In order to improve the understanding of the space composed by MMTS and OTS (MMTS+OTS), we compare different MMTS+OTS approaches. We use a feature model that we define in Section 3 and that we apply in Section 4, where more details about arrows 1 to 3 will follow. A discussion of some open issues are presented in Section 5. First, however, we continue with a definition of basic concepts from ontologies technologies and MDE in Section 2.

2. FOUNDATIONS

2.1 Ontological Technical Spaces

An ontology constitutes a formal conceptual model. Hence, its core concerns, i.e., formal definitions of classes and relationship, are germane to the software engineering community. Given their roots in knowledge representation and reasoning, however, ontologies have always been used differently than conceptual models in software and data engineering. Hence, the perspectives on modeling and using ontologies are slightly twisted if compared to conceptual models such as UML class diagrams.

2.1.1 Ontology Modeling

The process of modeling ontologies exhibits a couple of overlaps with the development of conceptual models [45]. Requirements elicitation is followed by a design phase when classes and relationships need to be defined similarly as in an UML class diagram. This stage, however, is followed by another step that depends on the ontology modeling paradigm

and its corresponding language.

In the realm of description logics-based ontologies [5], the strength of ontology modeling lies in disentangling conceptual hierarchies with an abundance of relationships of multiple generalization of classes (cf. [42]). For this purpose, description logics allows for *deriving* concept hierarchies from logically precisely defined class axioms, stating necessary *and* sufficient conditions of class membership.

In the realm of logic programming-based ontologies [2], the strength of ontology modeling lies in a formally integrated consideration of expressive class and rule definitions.

In both paradigms, the structure of class definitions may be validated by *introspecting* the model, using corresponding reasoning technology. In the first model of description logics, this is indeed the focus of its reasoning technology, while in the second model the focus of the corresponding reasoning technology is on reasoning with objects in a logical framework.

2.1.2 Ontology Languages and Reasoning

The language and reasoning paradigm that has been predominantly used and researched is the family of description logics languages including the W3C recommendation, the Web Ontology Language (OWL) [30]. All description logic languages allow for capturing the schema in the "terminological box" (T-Box) and the objects and their relationships in the "assertional box" (A-Box). The terminological box captures reasoning patterns restricted to knowledge about the class level, i.e., independent from a given situation, like the following:

Given	Every *service* has at least one *hasServiceDescription*.
Given	A *paymentFunction* has at least one *hasServiceDescription*. that is a *PaymentDescription*
Concluded	Every *paymentFunction* is a *service*.

In contrast, the A-Box captures reasoning patterns that concern knowledge about specific instances, exploiting a T-Box as above, i.e., knowledge about specific situations, like the following:

Given	*MyPaymentWay hasServiceDescription* "This service does this and this.".
Given	"This service does this and this.". is instance of *PaymentDescription*.
Concluded	*MyPaymentWay* is a *paymentFunction*.
Concluded	*MyPaymentWay* is a *service*.

The individual members of this family of languages differ in the set of modeling constructs they support. Depending on the exact configuration of allowed modeling primitives, a member of the family like DL Lite [13], the W3C recommendations OWL-lite and OWL-DL (two of the three versions of the Web Ontology Language), and KL-One belongs to the class of languages requiring polynomial, Exptime, Nexptime and undecidable sound and complete reasoning algorithms, respectively.

Ontology languages derived from logic programming are usually Turing powerful [29], but with their focus on instance reasoning act and work rather like deductive databases. This implies that they support closed world semantics (in contrast to the open world semantics of classical model theory

used in description logics and first-order predicate logics) where specific models are selected (i.e. least fix points according to an entailment operator). Recent research investigates the integration of the two semantic paradigms for joining description logics and logic programming [33].

Finally, RDF together with its associated schema language RDFS constitutes an inexpressive language for representing data and schema information in an exchangeable manner [49]. Its focus is on representing data located and linked on the Web rather than on expressive modeling of class definitions.

2.2 Metamodeling Technical Spaces

Whereas *models* describe a specific abstraction of reality (cf. e.g., [3]) *metamodels* define the modeling itself, including applied modeling technologies and modeling processes [11]. Today, the meta modeling space is associated with technologies developed in the UML environment [40]. Here, metamodeling is usually restricted to specify modeling (and programming) languages, only.

Metamodel-based approaches are based on a staged architecture of models and metamodels, where the structure of lower level models is explained by higher level metamodels. The Meta Object Facility [36] defines a four-layer structure, which is applied to define the UML and Domain Specific Languages. The top level (M3) defines the MOF itself. Language specifications like the UML specification are viewed as (linguistic) instances [4] of the MOF living on the metamodel level (M2). The model level (M1) contains concrete models, like class-diagrams or state machines, the notation of which is defined by metamodels on M2. Finally, M0 describes real world objects.

Models in the metamodel technical spaces are viewed as object networks or graphs. Their abstract syntax is usually defined by (UML-)class diagrams extended by constraint languages like OCL [38]. Furthermore, it is possible to extend the UML semantics with UML Profiles. The recently standardized semantics extension for ontology modeling is the Ontology Definition Meta model (ODM) [39], a family of UML metamodels defining various ontology languages, including OWL.

Technologies for dealing with object networks and graphs cover querying and transformation approaches. Requirements for transformation, querying and viewing languages within the MOF meta-modeling environment are proposed by OMG in the Query/View/Transform proposal [1].

Various transformation and querying approaches already exist or are being developed in consideration of QVT. Representatives of these approaches found their origins in graph querying or graph transformation like AGG [47] and GReQL [27]. Others, e.g., ATL [9], were directly developed according to QVT. A broad overview of transformation techniques is presented in [15].

In spite of their different purposes, OTS and MMTS share similar constructs. Recent work presents similarities between MOF and RDF [21], between OWL/RDF and Object-Oriented Languages [25] and between UML and OWL [39] [18]. We roughly summarize them in Table 1. For the subtleties, please refer to the cited papers.

Table 1: OTS and MMTS: comparable features

OTS	MMTS
ontology	package
class	class, classifier
individual and values	instance and attribute values
resource	model element
property	association, attribute
data types	data types
subclass, subproperty	subclass, generalization
enumeration	enumeration
domain, range	navigable, non-navigable
disjointness, union	disjointness, cover
cardinality	multiplicity

3. A FRAMEWORK FOR CLASSIFICATION

In this section, we present the results of applying a domain analysis to MMTS+OTS approaches. Domain analysis is concerned with analyzing and modeling the variabilities and commonalities of systems or concepts in a domain. The result of the analysis is a feature model described in this section. It comprises a feature diagram depicted in Fig. 2 using a Mind Map-like notation, a description of the features and their origin from specific technical spaces, and some examples. In Fig. 2, bold face features are mandatory and normal face features are optional. The mandatory features are included in all transformations between MMTS and OTSs presented in Section4. We presume that in the case of a mandatory feature with optional subfeatures, at least one of the subfeatures must be included.

The feature model reveals the different possible choices for a MMTS+OTS approach and can also be used as a taxonomy to categorize the existing approaches published in the literature as we do later in Section 4. The model does not intend to be normative, but rather informative.

3.1 Language

A language is defined based on:

1. A concrete syntax describing the way in which the language elements appear in a human-readable form. Extended BNF is frequently used to describe the concrete syntax of lexical notations. In the case of graphical notations, natural language and symbols are used to describe what graphical symbols represent the information, and how these symbols are laid out. A particular case of concrete syntax is a serialization syntax, which allows the language expressions to be made persistent or interchanged between tools. XML can be used as serialization syntax. Some syntactical variations may co-exist for one given language.

2. An abstract syntax portraying the elements that compose the language, and the possible combination of these elements. Abstract syntax graphs, metamodels or Extended BNF are commonly used to represent the abstract syntax of a language.

3. The semantics attributing meaning to the language primitives and the vocabulary. The attribution can be done by the means of a formal language using math-

Figure 2: Feature Model of Bridging OTSs and MDA MMTS.

ematics, or an informal language, like English. The relevant formal semantics for MMTS+OTS are:

- *Model-theoretic semantics*: assigns meaning to a set of logical sentences, i.e. a theory, by considering *all* possible interpretations that may be given to its atomic elements. Such a set of logical sentences is satisfiable if there is an interpretation that will render all the sentences to become true. Such an interpretation is called a model. If no model exists, the theory is inconsistent.

- *Fix-point semantics*: selects specific interpretations. Logical deduction is then considered true or false, depending not on all possible models, but based on the selected set of models.

- *Denotational semantics*: gives programs meaning by mapping them into mathematical objects. It presumes that programs and the objects they manipulate are symbolic realizations of abstract mathematical objects.

- *Axiomatic semantics*: is based on methods of logical deduction from predicate logic. The semantic meaning of a program is based on assertions about relationships that remain the same each time the program executes.

Another way of giving a semantics to a language is to translate expressions from one language into another language that already has a semantics.

The abstract syntax characterizes the primitives of a language. The concrete syntax realizes the primitives by a concrete notation. The semantics assigns meaning to the primitives and the models constructed using these primitives.

Let us consider three examples: UML is a modeling language with a graphical notation, an informal semantics described in natural language (there exist some translational semantics approaches for UML), uses a metamodeling approach to describe its abstract syntax, and natural language and symbols to describe the concrete syntax as well.

OWL-DL is an ontology modeling language with a lexical notation formalized by description logics. It is assigned a model-theoretic semantics, it is a subset of the formalism of first-order predicate logics, it has concrete and abstract syntax specified by Extended BNF, and it uses XML Schema to define a concrete serialization syntax.

RDF(S) is primarily a graph data model based on triples as abstract syntax graph, with a concrete lexical notation, a formal axiomatic semantics as well as a first-order logics description as formalization.

3.2 Formalism

Here, the term "formalism" is understood as formal language used to precisely define concepts of a model. A formalism is the basis for reasoning over models. We distinguish between four formalisms applicable to MMTS+OTS:

1. *First-Order Predicate Logics*: a logical language able to express relations between individuals using predicates and quantifiers.

2. *Description Logics*: a family of knowledge representation formalism aimed at unifying and giving a logical

basis to the well known traditions of Frame-based systems, Semantic Networks and KL-ONE-like languages, Object-Oriented representations, Semantic data models, and Type systems [5]. Core to each language from this family is its capability to express class definitions by restrictions on relationships to other classes and by inheritance relations. Though the exact expressiveness varies, all description logics languages are subsets of first-order predicate logics.

3. *Horn Rules*: Horn rules restrict First-Order Predicate Logics to axioms of a particular form. Though horn rules are in general Turing powerful, in practical situation it is often possible to oversee deductive consequences and to reason efficiently with terms (i.e., kind of objects). While horn rules can be given a model-theoretic semantics like First-Order Predicate Logics, in order to handle negation efficiently, most approaches come with some fix-point semantics in order to decide upon satisfiability (or inconsistency).

4. *Frame Logic*: is a syntactically more expressive variant of horn rules. It constitutes a deductive, object oriented database language, combining the declarative semantics and expressiveness of deductive database languages with the rich data modeling capabilities supported by the object oriented data model [2].

Ontologies and models written in a given language, e.g., OWL-DL, are usually translated to one or more formalisms, e.g., $\mathcal{SHOIN}$ [5], a member of the family of description logics languages, to realize reasoning. The UML class diagram can also be mapped to description logics, a corresponding example will be given in more detail in Section 4.1. We call this process "formalization".

3.3 Data Model

A data model is an underlying structure mandating how data is represented. The data model provide a basis for organizing the primitive elements of a language. This organization is used by the abstract syntax of the language to relate the primitives. We distinguish between four basic data models:

1. *Graph*: consists of (hyper-)edges and nodes.

2. *Tree*: constitutes a restricted graph data model having a hierarchical organization of the data.

3. *Object-based*: organizes data according to the object-oriented paradigm.

4. *Relational*: organizes data in relations.

A modeling approach can often be seen from the point of view of different data models. For instance, UML is commonly seen as a graph data model. In fact, however, UML class diagram instances could also be interpreted as a relational model.

OWL is primarily based on unary relations (i.e., logically-defined classes) and binary relations (i.e., relationships between objects), but there are alternative access methods, e.g., via Java object APIs or querying through the SPARQL, graph data model query language.

RDF(S) constitutes a graph data model, but it can also be seen as a kind of object model or a kind of constrained relational model.

3.4 Reasoning

Each type of reasoning is based on a formalism, typically a logical language, to deduce (infer) conclusions from a given set of facts (also called assertions) encoded in an model. Standard reasoning services include:

1. *Logical consistency*: checks whether a set of logical sentences, i.e., a logical theory, has an interpretation that makes all sentences become true, i.e., admits a model.

2. *Logical implication*: given a set of logical sentences as a premise (often called a "theory") another set of logical sentences may be implied as a conclusion because every model of the premise is also a model of the conclusion.

3. *Subsumption*: is a special case of checking logical implications. Subsumption tests whether one class definition is more specific than another one — given a set of logical sentences as background theory. Subsumption tests can be used to generate a sound and complete classification of a set of class definitions.

4. *Equivalence*: applies subsumption tests in two directions in order to determine equivalence between two class definitions.

5. *Extension test*: an extension test checks whether a tuple is contained in a logical relation. Specifically, checking of instantiation tests whether an instance belongs to the extension of a class, which is a unary relation.

All standard reasoning services in first-order predicate logics (and in description logics, specifically) illustrated here can be based on consistency checking. In horn rules formalisms, reasoning is rather defined either based on resolution or based on naïve bottom-up evaluation.

3.5 Querying

Querying, accompanied by transformation, plays an important role for accessing and bridging between technical spaces. The work by Haase et al. comparing aspects of query languages for ontologies have been used to identify features of querying [22]:

1. *Inference support*: when information is retrieved from a technical space a query engine may access only explicitly available data (e.g., SPARQL [41]) or it may include facts derived by using a reasoning (e.g., OWL-QL [19] or SAIQL [26]).

2. *Closure*: a query language may represent the results of a query on a model (i.e., a kind of database) either in the same format as the model itself or in a different paradigm. For instance, the earliest RDF query languages returned results as variable bindings, i.e., as relations rather than graphs, while SPARQL may return results in its native paradigm, i.e., as a graph.

3. *Safety*: a query language is considered safe, whenever a syntactically correct query returns a finite set of results.

Queries are expressed in a language over a data model in a modeling level, and can use a reasoning service. For example, OCL can be used as a query language with lexical notation over an UML object data model. SPARQL[41] is a query language with lexical notation over RDF graph data model without reasoning support (according to the specification; some implementation still support reasoning) and with results being either represented as relations or as graphs.

3.6 Rules

Rules are present inside technical spaces as well as in transformations between them. Rule languages can be considered to include a querying mechanism over a data model. The term "rules" is ambiguous and includes in its range:

1. *Integrity constraints*: restrict the number of possible interpretations, but they do not add inferences and just signal exceptions.

2. *Derivation rules*: comprise conditions from which a fact is derived as a conclusion whenever the rule holds.

3. *Reaction rules*: comprise a triggering event and a condition that leads to a triggered action whenever a triggered condition holds.

4. *Transformation rules*: process models or data from one source to one target. They consist of conditions and a transformation function, which returns its result whenever the condition holds.

5. *Logical rules*: describe a logical axiom that holds.

For example, OCL can be used to write integrity constraints and derivation rules as well. Part from the UML specification called Action Semantics could be used to specify reaction rules. A SQL Trigger is another example of reaction rule.

F-Logic rules are logical rules that can be considered to constitute derivation rules and that can be configured to model integrity constraints. TRIPLE [16] is a reasoner for which a hybrid rule language with lexical notation for querying and translating RDF models has been designed. It is primarily based on logic programming and has strong ties with F-Logic. DL-safe rules [34] is a logical rule mechanism extending OWL-DL (aside for some minor primitives of OWL-DL) while allowing for sound and complete reasoning with class definitions and a restricted rule language that defines specific logical axioms.

ATL [9] and QVT [37] are languages with lexical notation, metamodeling abstract syntax and can be used to write transformation rules.

3.7 Transformation

A transformation definition is a set of transformation rules that together describe the conversion of one model in the source language into another related model in the target language [24]. A transformation definition uses querying over a data model and transformation rules to manipulate the source and target metamodels, and can involves reasoning. Concerning MMTS+OTS, we distinguish between four aspects of transformations:

1. *Semantical*: The semantical aspect of a transformation differs between *precise* transformation or *approximating* transformations. Approximating transformations give up on soundness (rarely) or completeness (more often) in order to speed up subsequent querying or reasoning. Precise transformations do not change the querying and/or reasoning results.

2. *Syntactical*: this aspect involves *graph-pattern*, which draws on the theoretical work on graph transformations operating on typed, attributed, labeled graphs, and *executable logic* of the transformation, which can be declarative or imperative.

3. *Directionality*: concerns the ability to generate models in different directions based on the definition of a transformation. Bidirectional transformations are sufficient to transform forward and backward between source and target models. Examples include QVT and UMLX. Unidirectional transformation allow for transformations in exactly one direction, such as ATL, in general.

For example, UMLX [50] is a transformation language with graphical notation, declarative executable logic, precise semantics, and can produce bidirectional model transformations. QVT [37] and ATL[9] are languages with lexical notation, realize precise transformations, and can use declarative and imperative executable logic. They both allow for defining transformation rules, using OCL as query language over UML object models.

3.8 Mediation

Mediation is the process of reconciling differences between heterogeneous models. It plays a central role in MMTS+OTS, as models in different languages must coexist. Features of mediation are:

1. *Mapping*: the declarative specification of the correspondences between different elements of the two models. In a transformation process, the mapping specification precedes the transformation definition.

2. *Integration*: focuses on interoperability between models so that they work together effectively. It comprises:

   - Aligning: preserves the source models and produces a new model containing additional axioms to describe the relationship between the concepts from the source models.

   - Merging: creates one new merged model based on concepts found in the source models.

3. *Composition*: comprises the combination of elements that accord to overlapping concepts in different source models. Usually, each source model handles a different dimension of the overlapping elements. A weaving process does not necessarily produce a merge, but it can produce a model in a third language with new knowledge based on the source models.

Both integration and composition make use of mappings to specify overlaps.

For example, an aligning between a UML class diagram and an ontology could involve the specification of the mappings between both models. The resulting model can be queried independently whether the queried class resides in the UML class diagram or in the ontology.

A merging involves the generation of a completely new model in one of the given languages. Both integration strategies consider relating only the similar primitives of each language. A composition can generate a model in a new language with the expressiveness of the previous languages, considering particularly a union of the primitives of each language. For example, a composed model may allow an ontological class definition while having the specification of a class method on the UML side.

3.9 Modeling Level

In Model Driven Engineering, the concepts described by models are organized according to different levels. The concepts of a M level define the concepts of $M-1$ Level. Such as organization is defined by [8] as follows:

0. System: the executable system, the runtime instances.

1. Model: a representation of the system.

2. metamodel: defines the concepts in which the model is defined.

3. Metametamodel: defines the core concepts in which the metamodel and itself are defined.

This organization corresponds to the OMG's Four layered metamodel architecture: the metametamodel level (M3), the metamodel level (M2), the model level (M1) and the run-time instances (M0). The modeling levels are described using a language and are organized according to a data model.

4. TRANSFORMING BETWEEN MMTS AND OTS

In this section, we describe some example transformations between meta modeling and ontology-based approaches. We have formed four groups of approaches and within each group we delve into some details of a "prototypical" approach, which is also classified according to the framework defined in Section 3. The reader may note that the variation is indeed larger than can be fully covered by this description.

4.1 Model Checking

This category groups the works that use automated reasoning techniques for checking and validation of models in formalized languages. Most reasoning approaches for validation check some specification against some design. The description logics technical spaces, however, have specifically been defined to validate the *internal* consistency of a set of class definitions. To exploit this model of validation, one may transform a part of a given MDE-based model, e.g., the class diagram of a PIM based on UML, into a set of OWL class definitions (cf. arrow 1 in Fig 1; cf. [7]) and one may check class hierarchy relationships, property hierarchies as well as the logical consistency of instantiating classes.

We illustrate this process by using the simple UML class diagram of university accounts depicted in Fig. 3. The diagram shows that a `WebPortalAccount` is a particular kind of `UserAccount` and that each `UserAccount` is owned by one and only one `User`. Additionally, a `User` can be only of two different kinds, a `Researcher` or a `Student`. A `Researcher` can have only one `WebPortalAccount`. The association class `Uses` specializes the association class `Owns`.
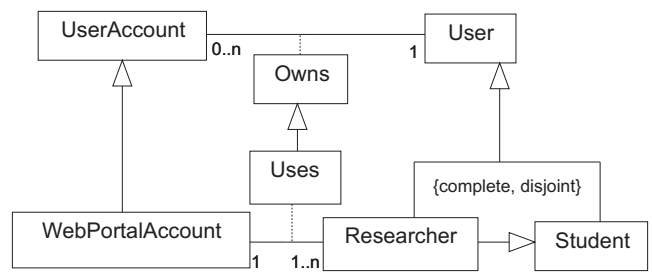

Figure 3: Checking consistency of UML models.

After applying the transformation from UML into a description logics model, such as OWL (more specifically,[7] mapped it into $\mathcal{ALCQI}$), we ask the reasoner to verify the model. By reasoning over such a diagram, we discover some undesirable properties. First of all, the class `Researcher` must be empty and, hence, cannot be instantiated. The reason is that the disjointness constraint asserts that there is no `Researcher` who is also `Student`. Furthermore, since the class `User` is made up by the union of classes `Researcher` and `Student`, and since `Researcher` is empty, the classes `User` and `Student` are equivalent, implying redundancy.

By dropping the generalization `Student-Researcher`, we arrive at a valid model. If we invoke the reasoner one more time, we can refine the multiplicity of the role `Researcher` in the association `uses` to 1. `Owns` is a generalization of `Uses`, hence every link of `Uses` is a link of `Owns`, since every `Account` is owned by exactly one `User`, necessarily every `WebPortalAccount` is used by at most one `Researcher`, since `WebPortalAccount` is a subclass of `Account`.

Reconsidering our feature model depicted in Fig. 2, the configuration of this category uses the following features: (i) a model at a given modeling level (model, metamodel or metametamodel), written in a given language, using an object data model; (ii) a target model, written in a given language with model-theoretic semantics, including one formalism, reasoning capability, querying with closure, inference support and safe; (iii) a mapping specification describing the links between the models; (iv) a unidirectional, declarative and precise transformation definition, which includes transformation rules and querying.

Another work, that roughly fits into this category of MMTS to OTS transformation for the purpose of model checking, is [46]. It proposes an approach to detect and resolve inconsistencies between different versions of UML models specified as a collection of class diagrams, sequence diagrams and state diagrams. It presents a Domain Specific Language (UML Profile) able to describe the evolution of the models.

4.2 Model Enrichment

Model enrichment aims to use ontologies for enrichment of models from MMTS. One example is given by arrow 3 in Fig. 1. The PIM specified in MMTS is transformed to a TRIPLE model from OTS. By reasoning on the TRIPLE model one may derive new facts to be transformed back to and included in MMTS, e.g. in the PIM.

Figure 4 illustrates a simplifying example: (Step 0: Model) It depicts two models capturing bibliographical references. On the left side, the model `Ma` comprises the class `Publication`, which generalizes `Article` and `Thesis`,
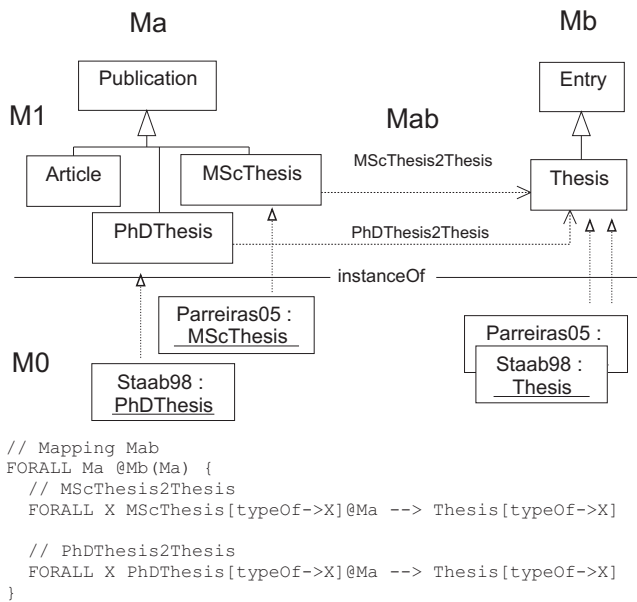
```
// Mapping Mab
FORALL Ma @Mb(Ma) {
  // MScThesis2Thesis
  FORALL X MScThesis[typeOf->X]@Ma --> Thesis[typeOf->X]

  // PhDThesis2Thesis
  FORALL X PhDThesis[typeOf->X]@Ma --> Thesis[typeOf->X]
}
```

Figure 4: Mapping between two models Ma and Mb.

which generalizes `MScThesis` and `PhDThesis`. On the right side, the model `Mb` includes the classes `Entry` and `Thesis`. In the middle, there is a mapping model, `Mab`, with a link `MScThesis2Thesis` mapping `MScThesis` onto `Thesis` and a link `PhDThesis2Thesis` mapping a `PhDThesis` onto a `Thesis`.

(Step 1: Forward Transformation) The three models may be transformed into the TRIPLE language from the OTS. The resulting TRIPLE model includes all instances and classes of `Ma` and `Mb` as RDF and furthermore it contains the two logical rules depicted in the lower part of Fig. 4.

(Step 2: Logical Querying) The logical rules allow for querying of `Ma` instances through the view of `Mb` in the TRIPLE OTS [16]. The corresponding query is defined in TRIPLE by `FORALL X,Y,Z <- X[Y->Z]@Mb(Ma)`. These retrieved instance triples may be added as triples to the RDF space of the `Mb` part of the TRIPLE OTS.

(Step 3: Backward Transformation) Eventually, one may transform the latter results back to `Mb`, then including all the objects of `Ma` as seen through `Mb` in `Mb`.

While the given example is too simple to be of use in the software engineering process, real applications may exploit the TRIPLE inferencing and enrichment *(i)* to translate (database) objects between PSM/code models at runtime, or *(ii)* to perform more complex reflections (i.e. at the model level) that need the help of logic programming, e.g. recursive logical rules such as exploited in [35].

Regarding the feature model, the configuration of features include: (i) a model at a given modeling level (model, metamodel or metametamodel), written in a given language, using an object data model; (ii) a target model, written in a given logical language, here with fix-point semantics, reasoning capability, querying with closure, inference support and safety; (iii) a mapping specification describing the links between the models; (iv) a *bidirectional* declarative transformation definition, which includes transformation rules and querying; (v) logical rules and reasoning to make the knowl-
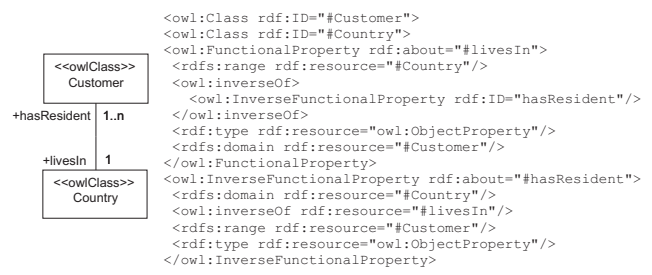

Figure 5: Ontology Modeling with UML Profile.

edge explicit on the OTS side.

As this is not a closely grouped category, further work in this category exhibit strongly varying facets. Billig et al. [10] use TRIPLE to generate mappings between a PIM and a PSM describing user requirements as input. It comprises a transformation from MMTS into OTS (TRIPLE), the generation of the mappings, the transformation into a PSM under OTS and the transformation OTS to MMTS of the PSM. Roser and Bauer [43] present a framework to automatically generate model transformations between MMTS models using the OTS; Kappel et al.[23] provide an approach for model-based tool integration. It consists of transforming two metamodels from MMTS into an OTS, uses reasoning services and generates mappings between the two models represented in the OTS.

4.3 Ontology Modeling

This category groups the efforts to give a graphical notation to ontology modeling (arrow 2 in Fig. 1). Referring to our feature model, the configuration of this category includes: (i) a model written in a given language with graphical notation from MMTS; (ii) a target model written in a given language and including one formalism from OTS; (iii) a mapping specification describing the links between the models; (iv) a bidirectional and precise transformation definition, which includes transformation rules and querying. It needs not involve reasoning per se.

Figure 5 illustrates this configuration. An UML Profile for ontology modeling is used to stereotype the classes `Customer` and `Country` as `OWLClass`. The `objectProperty`s are represented as roles in the association. This UML model is then translated to OWL serialization syntax, using a transformation definition.

Cranefield and Purvis [14] and Falkovych et al.[18] introduced UML without extensions as Ontology Representation Language capable of representing ontologies.

Extensions of the Unified Modeling Language for ontology development were proposed [6], culminating in a new metamodel into the MDA family of modeling languages — the Ontology Definition Metamodel [12] [39] [17]. These approaches use a DSL (UML Profile) to represent the ontology, a mapping onto the ODM, a transformation from the ODM into the serialization syntax of the OWL ontology language.

4.4 Hybrid Approaches

This category embraces our attempt at considering behaviorial and representational aspects of modeling an application at MMTS (arrows 2+a+b in Fig. 1), called TwoUse (Transforming and Weaving Ontologies and UML in Software Engineering) [44]. It involves: (i) a model written in

profiled UML with OCL expressions; (ii) a target model, written in OWL with model-theoretic semantics and Description Logics formalism, and; (iii) a *composition* including a mapping specification describing the links between the models; (iv) a bidirectional and precise transformation definition, which includes transformation rules and querying. It differs from the former category as there is not one target model, but rather the aim is to eventually have models for code *as well as* for a logical space queried during runtime.

Consider the example of Subsection 4.3: the modeler, using a UML Profile for ontology, designs only an ontology, but cannot design an object oriented model in *the same diagram*, i.e., he cannot add the method `getGermanCustomers()` to class `Customer`, because `Customer` is not an object oriented class, it is an ontology class. With TwoUse, the modeler uses OCL expressions to describe the query operation in the same diagram, i.e, the class `Customer` becomes *also* an object oriented class. Moreover, this operation can query the ontology, i.e., invoke a reasoning service at runtime that uses the same ontology.

The ontology can be directly generated from the model (PIM)(arrow 2 in Fig. 1), whereas the object oriented classes and OCL expressions are translated into a specific platform (arrow b) and later into Java code (arrow b) including the API for ontology and reasoning invocation.

5. DISCUSSION AND OUTLOOK

We have illustrated variations on the principle idea of using meta modeling technical space (MMTS) with different ontological technical spaces (OTSs). The basic patterns we find in our own work as well as in related works is that next to existing technical spaces of established meta modeling frameworks, new technical spaces are positioned that either enrich or exploit the software engineering capabilities by or for ontology technologies. We have identified the main characteristics of such approaches and designed a feature model to enlighten the possible conceptual choices. We have applied our model, illustrating the use of different ontology technologies.

Such positioning of technical spaces has been made possible by intensive research work in the last 10 years in the fields of MDE and ontology technologies. There remains a considerable body of problems being currently tackled or opened for future work:

First, *scaleable ontological reasoning technology* has matured tremendously over the last 10 years and current ideas and implementations let us expect that future reasoners will scale to higher efficiency by one or several orders of magnitude. Nevertheless, some reasoning tasks will remain computation-intensive and will not be exploitable for interactive design, but rather in batch validation processes. Techniques for semantic transformations between different ontology reasoning techniques will be required and that the software developer benefits from the most appropriate and most efficient technique at each given point in the software development process.

Second, current models in MDE still assume rigidly defined class hierarchies. In the future, we expect more interactions between flexible logical models of expressive class definitions being used and hierarchically ordered at runtime and software engineering practices exploiting this flexibility and logical rigor for sound definition and management of complex software systems.

Third, in the past the semantics of ontology technologies like OWL and modeling approaches such as MDA were considered disjoint, as the former was based on open-world reasoning and the latter on closed world reasoning. More recent work shows that there is a common logical basis that let us exploit joint modeling with either paradigm — using both point of views at the same time (cf. [32]).

Hence, in the future, we expect still much more light onto the dimly lit field of MMTS and OTS — towards more benefits for either discipline.

6. REFERENCES

[1] MOF 2.0 Query/Views/Transformations RFP, October 2002.

[2] J. Angele and G. Lausen. Ontologies in F-logic. In *Handbook on Ontologies*, pages 29–50. 2004.

[3] L. Apostel. Towards the formal study of models in a non formal science. *Synthese*, 12:125–161, 1960.

[4] C. Atkinson and T. Kühne. Model-Driven Development: A Metamodeling Foundation. *IEEE Software*, (5):36–41, September/October 2003.

[5] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.

[6] K. Baclawski, M. M. Kokar, P. A. Kogut, L. Hart, J. E. Smith, J. Letkowski, and P. Emery. Extending the unified modeling language for ontology development. *Software and System Modeling*, 1(2):142–156, 2002.

[7] D. Berardi, D. Calvanese, and G. D. Giacomo. Reasoning on UML class diagrams. *Artif. Intell.*, 168(1):70–118, 2005.

[8] J. Bézivin. On the unification power of models. *Software and System Modeling*, 4(2):171–188, 2005.

[9] J. Bézivin, G. Dupé, F. Jouault, G. Pitette, and J. Rougui. First experiments with the ATL Model transformation language: Transforming XSLT into XQuery. In *OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture*. 2003.

[10] A. Billig, S. Busse, A. Leicher, and J. G. Süss. Platform independent model transformation based on TRIPLE. In *Middleware '04*, pages 493–511. Springer, 2004.

[11] S. Brinkkemper. Method Engineering: Engineering of Information Systems Developement Methods and Tools. *Information & Software Technology*, 38(4):275–280, 1996.

[12] S. Brockmans, R. Volz, A. Eberhart, and P. Löffler. Visual modeling of OWL DL ontologies using UML. In *Proc. of ISWC 2004*, pages 198–213, 2004.

[13] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. DL-lite: Tractable description logics for ontologies. In *Proc. of AAAI 2005*, pages 602–607, 2005.

[14] S. Cranefield and M. K. Purvis. UML as an ontology modelling language. In *Intelligent Information Integration*, volume 23 of *CEUR Workshop Proceedings*, 1999.

[15] K. Czarnecki and S. Helsen. Classification of model transformation approaches. *Proceedings of the 2nd*

OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture, 2003.

[16] S. Decker, M. Sintek, A. Billig, N. Henze, P. Dolog, W. Nejdl, A. Harth, A. Leicher, S. Busse, J. L. Ambite, M. Weathers, G. Neumann, and U. Zdun. TRIPLE - an RDF rule language with context and use cases. In *Rule Languages for Interoperability*, 2005.

[17] D. Djurić, D. Gašević, V. Devedžić, and V. Damjanovic. A UML profile for OWL ontologies. In *MDAFA*, pages 204–219, 2004.

[18] K. Falkovych, M. Sabou, and H. Stuckenschmidt. Uml for the semantic web: Transformation-based approaches. In *Knowledge Transformation for the Semantic Web*, pages 92–106. 2003.

[19] R. Fikes, P. Hayes, and I. Horrocks. OWL-QL: A language for deductive query answering on the semantic web. Technical Report KSL 03-14, Stanford University, Stanford, CA, 2003.

[20] D. Frankel, P. Hayes, E. Kendall, and D. McGuinness. The model driven semantic web. In *1st International Workshop on the Model-Driven Semantic Web (MDSW2004)*, Monterey, California, USA, 2004.

[21] D. Gašević, D. Djurić, and V. Devedžić. MDA-based automatic OWL ontology development. *Int. J. Softw. Tools Technol. Transf.*, 9(2):103–117, 2007.

[22] P. Haase, J. Broekstra, A. Eberhart, and R. Volz. A comparison of rdf query languages. In *Proc. of ISWC 2004*, Hiroshima, Japan, 2004.

[23] G. Kappel, E. Kapsammer, H. Kargl, G. Kramler, T. Reiter, W. Retschitzegger, W. Schwinger, and M. Wimmer. Lifting metamodels to ontologies: A step to the semantic integration of modeling languages. In *Proc. of MoDELS 2006*, volume 4199 of *LNCS*, pages 528–542. Springer, 2006.

[24] A. G. Kleppe, J. B. Warmer, and W. Bast. *MDA Explained, The Model Driven Architecture: Practice and Promise.* Addison-Wesley, Boston, 2002.

[25] H. Knublauch, D. Oberle, P. Tetlow, and E. Wallace. A semantic web primer for object-oriented software developers. W3c working group note, W3C, Mar. 2006.

[26] A. Kubias, S. Schenk, S. Staab, and J. Z. Pan. OWL SAIQL - an OWL DL query language for ontology extraction. In *Proc. of OWLED-07*, 2007.

[27] B. Kullbach and A. Winter. Querying as an Enabling Technology in Software Reengineering. In *Proc. of CSMR 1999.* IEEE Computer Society.

[28] I. Kurtev, J. Bézivin, and M. Aksit. Technological spaces: An initial appraisal. In *CoopIS, DOA'2002 Federated Conferences, Industrial track*, Irvine, 2002.

[29] J. W. Lloyd. *Foundations of logic programming; (2nd extended ed.).* Springer, New York, NY, USA, 1987.

[30] D. L. Mcguinness and F. van Harmelen. OWL web ontology language overview, February 2004.

[31] J. Miller and J. Mukerji. Mda guide version 1.0.1. Technical report, OMG, 2003.

[32] B. Motik, I. Horrocks, and U. Sattler. Bridging the gap between OWL and relational databases. In *Proc. of WWW '07*, pages 807–816. ACM Press, 2007.

[33] B. Motik and R. Rosati. A faithful integration of description logics with logic programming. In M. M. Veloso, editor, *IJCAI*, pages 477–482, 2007.

[34] B. Motik, U. Sattler, and R. Studer. Query answering for owl-dl with rules. *Journal of Web Semantics*, 3(1):41–60, 2005.

[35] D. Oberle, A. Eberhart, S. Staab, and R. Volz. Developing and managing software components in an ontology-based application server. In *Proc. of Middleware-04*, pages 459–477, 2004.

[36] OMG. *Meta Object Facility (MOF) Specification.* Object Modeling Group, 2000.

[37] OMG. *MOF QVT Final Adopted Specification.* Object Modeling Group, June 2005.

[38] OMG. *Object Constraint Language Specification, version 2.0.* Object Modeling Group, June 2005.

[39] OMG. *Ontology Definition Metamodel.* Object Modeling Group, August 2005.

[40] OMG. *Unified Modeling Language: Superstructure, version 2.1.1.* Object Modeling Group, February 2007.

[41] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF (working draft). Technical report, W3C, March 2007.

[42] A. L. Rector, N. Drummond, M. Horridge, J. Rogers, H. Knublauch, R. Stevens, H. Wang, and C. Wroe. OWL pizzas: Practical experience of teaching OWL-DL: Common errors & common patterns. In *Proc. of EKAW 2004*, pages 63–81, 2004.

[43] S. Roser and B. Bauer. An approach to automatically generated model transformations using ontology engineering space. In *Proceedings of Workshop on Semantic Web Enabled Software Engineering (SWESE)*, Athens, GA, U.S.A., 2006.

[44] F. Silva Parreiras, S. Staab, and A. Winter. TwoUse: Integrating UML models and OWL ontologies. Technical Report 16/2007, Universität Koblenz-Landau, Fachbereich Informatik, 4 2007.

[45] S. Staab, R. Studer, H.-P. Schnurr, and Y. Sure. Knowledge processes and ontologies. *IEEE Intelligent Systems*, 16(1):26–34, 2001.

[46] R. V. D. Straeten, T. Mens, J. Simmonds, and V. Jonckers. Using description logic to maintain consistency between UML models. In *Proc. of UML 2003*, volume 2863 of *LNCS*, pages 326–340. Springer, 2003.

[47] G. Taentzer. AGG: A graph transformation environment for modeling and validation of software. In *Second International Workshop AGTIVE 2003*, volume 3062 of *LNCS*, pages 446 – 453. Springer, 2004.

[48] P. Tetlow, J. Z. Pan, D. Oberle, E. Wallace, M. Uschold, and E. Kendall. Ontology driven architectures and potential uses of the semantic web in systems and software engineering. W3C Working Draft Working Group Note 2006/02/11, W3C, 03 2006.

[49] W3C. Resource description framework (rdf). http://www.w3.org/RDF/, 2004.

[50] E. D. Willink. UMLX: A graphical transformation language for MDA. In A. Rensink, editor, *TR-CTIT-03-27*, pages 13–24, Enschede, 2003.