An Ontology for Software Models and Its Practical Implications for Semantic Web Reasoning*

Matthias Bräuer and Henrik Lochmann

SAP Research CEC Dresden
Chemnitzer Str. 48, 01187 Dresden, Germany
{matthias.braeuer,henrik.lochmann}@sap.com

Abstract. Ontology-Driven Software Development (ODSD) advocates using ontologies for capturing knowledge about a software system at development time. So far, ODSD approaches have mainly focused on the unambiguous representation of domain models during the system analysis phase. However, the design and implementation phases can equally benefit from the logical foundations and reasoning facilities provided by the Ontology technological space. This applies in particular to Model-Driven Software Development (MDSD) which employs models as first class entities throughout the entire software development process. We are currently developing a tool suite called HybridMDSD that leverages Semantic Web technologies to integrate different domain-specific modeling languages based on their ontological foundations. To this end, we have defined a new upper ontology for software models that complements existing work in conceptual and business modeling. This paper describes the structure and axiomatization of our ontology and its underlying conceptualization. Further, we report on the experiences gained with validating the integrity and consistency of software models using a Semantic Web reasoning architecture. We illustrate practical solutions to the implementation challenges arising from the open-world assumption in OWL and lack of nonmonotonic queries in SWRL.

1 Introduction

In recent years, researchers and practitioners alike have started to explore several new directions in software engineering to battle the increasing complexity and rapid rate of change in modern systems development. Among these new paradigms is *Semantic Web Enabled Software Engineering (SWESE)*, which tries to apply Semantic Web technologies (such as ontologies and reasoners) in mainstream software engineering. This way, SWESE hopes to provide stronger logical foundations and precise semantics for software models and other development artifacts.

* This work is supported by the feasiPLe project (partly financed by the German Ministry of Education and Research (BMBF)) and by Prof. Assman from Chair of Software Technology at Dresden University of Technology.

S. Bechhofer et al.(Eds.): ESWC 2008, LNCS 5021, pp. 34–48, 2008.

The application of ontologies and Semantic Web technologies in software engineering can be classified along two dimensions [13]: the kind of knowledge modeled by the ontology and whether the approach tackles runtime or development time scenarios. If ontologies are employed during development time to capture knowledge of the software system itself (rather than the development infrastructure or process), we speak of *Ontology-Driven Software Development (ODSD)*.

So far, most approaches to ODSD have focused on using ontologies as unambiguous representations of domain (or conceptual) models during the initial phases of the software engineering process, i.e., requirements engineering and systems analysis. However, the design and implementation phases can equally benefit from the logical foundations and reasoning facilities provided by the Semantic Web. This applies in particular to *Model-Driven Software Development (MDSD)*, a development paradigm that employs models as first class entities throughout the entire development process [27].

MDSD advocates modeling different views on a system (e.g., data entities, processes, or user interfaces) using multiple domain-specific modeling languages (DSLs) [14]. This raises the need for sophisticated consistency checking between the individual models, decoupling of code generators, and automatic generation of model transformations [2]. We are currently developing a toolsuite called *HybridMDSD* that aims at leveraging Semantic Web technologies to address these challenges.

In this paper, we introduce HybridMDSD with its ontological foundation, an upper ontology for software models. Additionally, we concentrate on the challenges that arise from using Semantic Web technologies to validate the integrity and consistency of multiple software models. Our main contribution is an analysis of practical solutions to the implementation challenges posed by the open-world assumption in OWL and lack of nonmonotonic queries in SWRL. By doing so, we highlight the need for nonmonotonic extensions to the Semantic Web languages in the context of Ontology-Driven Software Development.

After briefly reviewing selected aspects related to ODSD in Sect. 2, we present our approach and its benefits in Sect. 3. Section 4 describes the main concepts of our newly developed upper ontology. Based on its axiomatization, Sect. 5 elaborates in detail on the challenges and solutions for Semantic Web reasoning over closed-world software models. In Sect. 6, we briefly highlight related work to place our method in context. Section 7 concludes on the paper.

2 Ontologies and Models

This section reviews the prevalent view on ontologies and software models in ODSD and the relation to the Semantic Web. This provides the foundation for highlighting the differences of our approach and discussing the resulting challenges in the following sections.

So far, ontology-driven software development has focused on using ontologies for domain representation in conceptual and business modeling [12]. This appears

to stem from the view that ontologies and software models have differing, even opposing intentions. For instance, in a recent proposal to unify *Model-Driven Architecture (MDA)* and Semantic Web technologies [1], the authors point out that both ontologies and models are means to describe a domain of interest by capturing all relevant concepts and their relationships. However, a key difference between the two approaches is that a model is a *prescriptive* representation of a particular domain under *closed-world assumption (CWA)*. Essentially, this means that everything that is not explicitly contained in the model does not exist. Thus, models are ideally suited as exact specifications for software systems. Ontologies, by contrast, are *descriptive* and possibly incomplete representations of the "real world". They follow the *open-world assumption (OWA)* which means that a statement is not necessarily false if it cannot be proved true.

The closed-world assumption in software models is closely related to *nonmonotonic reasoning*. Nonmonotonic logics allow to make decisions based on incomplete knowledge, causing previously-drawn conclusions to be retracted when new information becomes available. An important property of nonmonotonic logics is strong negation or *negation as failure*, which allows to infer $\neg P$ if P cannot be proved. This is vital to validate integrity constraints in closed-world data models, but can cause problems in the open world of the Semantic Web. Since incomplete or changing knowledge is common in the web, nonmonotonicity could often lead to wrong conclusions. Also, implementing efficient nonmonotonic reasoners is difficult, because predicate nonmonotonic logic is undecidable. As a result, the Semantic Web is currently built on monotonic formalisms: the Web Ontology Language (OWL) corresponds to description logics, while the Semantic Web Rule Language (SWRL) is based on Horn clause rules.

3 HybridMDSD

Model-Driven Software Development facilitates the generation of executable software assets from technical abstractions of concrete domain knowledge. However, applying multiple domain-specific models to describe different views on the same system is still challenging, because it remains difficult to properly describe the semantic references and interdependencies between elements in different models and to maintain consistency. The HybridMDSD project tries to alleviate these challenges. In the following, we outline our approach and highlight its benefits for MDSD.

3.1 Approach

The core idea of our approach is to capture pure system-related knowledge of modeling languages and actively use this knowledge during language instantiation. To implement this, we establish a binding between the constructs of a modeling language and the concepts and relationships of an axiomatized upper ontology [20,4]. During system modeling, this binding is used to construct an ontology knowledge base that contains information about elements from different models and their interdependencies. We call this knowledge base a *semantic*

connector, because it allows to create semantically sound references between different domain-specific models in a holistic view of the entire system.

3.2 Benefits

Mapping constructs of different modeling languages to a single reference ontology allows the generation of model transformations. Additionally, the semantic connector forms the basis for comprehensive ABox reasoning over the set of all semantically connected model elements. Thus, it permits integrity and consistency validation across the boundaries of individual models (cf. Sect. 5). This enables several modelers to work collaboratively on different models of a large system while maintaining semantic consistency between the individual views. In addition, domain-specific inference rules facilitate automatic adaptation of model instances in case of modifications to one or more connected models. Such modifications commonly occur during software evolution and especially in Software Product Line Engineering (SPLE), where a variable product feature may affect several system models [20].

4 An Ontology for Software Models

This section introduces the *Unified Software Modeling Ontology (USMO)*, a new upper ontology that is the basis for the semantic connector in the HybridMDSD project. USMO is the result of a careful analysis of existing foundational ontologies in the area of conceptual modeling, such as the *Bunge-Wand-Weber (BWW)* ontology [8], the *General Foundational Ontology (GFO)* [11] and the *Unified Foundational Ontology (UFO)* [12]. Specifically, we have compared the ontological foundations of conceptual models with those of software models. Our study revealed major differences in the corresponding ontological interpretation, which eventually prompted us to derive our own upper ontology. We have successfully employed this new ontology in a case study that involved the semantic integration of several domain-specific models [5]. A detailed description of all USMO concepts and relationships is beyond the scope of this paper, so we limit the discussion to those elements relevant in the following sections. A comprehensive coverage of the entire ontology — including its philosophical background and conceptualization of physical objects changing over time — can be found in [5].

4.1 Concepts and Relationships

In its current version, USMO totals 27 classes and 56 properties. In line with UFO, we divide the world of things into Entitys and Sets. In the context of software modeling, Entitys simply correspond to model elements. At the most general level, the Entity class therefore represents language constructs in modeling languages. Figure 1 depicts the top level concepts of USMO.

In our ontology, an Entity is simultaneously classified as (1) either an Actuality or a Temporality *and* (2) either a Universal or a Particular. The Actuality and

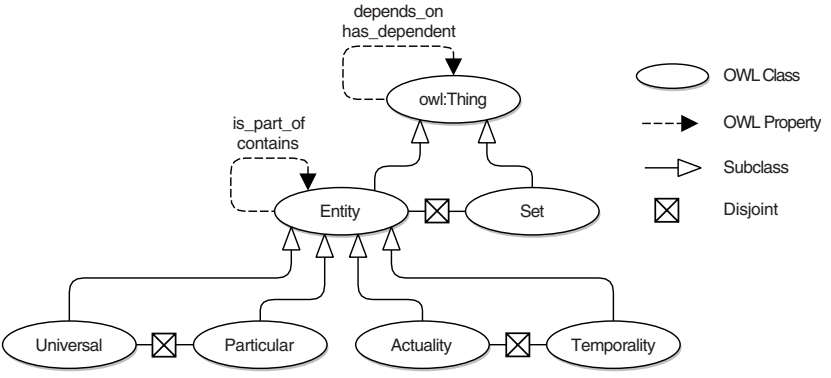

Fig. 1. Elementary concepts and relationships

Temporality concepts represent the philosophical notions of Endurant and Perdurant, respectively [9]. While an Actuality is wholly present at any time instant, Temporalitys are said to "happen in time". In the context of software models, this allows to ontologically interpret both structural and temporal elements.

Universals represent intensional entities that can be instantiated by Particulars. This distinction facilitates the ontological interpretation of both *type models* (e.g., class diagrams) and *token models* (e.g., object diagrams and behavioral models) [19]. Figure 2 illustrates the relationships between a Universal and its *extension*, which is the Set of instantiating Particulars. Note that USMO only supports two levels of ontological instantiation, so a Universal cannot instantiate other Universals.

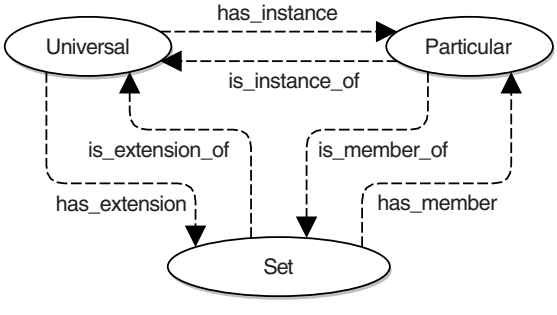

Fig. 2. The model of ontological instantiation

At the top level, we define two major relationships. The most important one is *existential dependency*, since the corresponding properties depends_on and has_dependent subsume all other USMO properties. The dependency relation therefore spans a directed, acyclic graph over all model elements captured in the semantic connector knowledge base. With the help of a reasoner, this design

allows complex impact analysis across several domain-specific models. The second important relationship is *containment*, which adds the notion of transitive, non-shareable ownership.

To conclude the overview of our ontology, Fig. 3 shows the USMO conceptualization of universal structural elements, which are typically found in type models. A Schema is an Actuality Universal that classifies independent entities. By contrast, a Property cannot exist on its own and always depends on at least one Schema. There are two types of Propertys: a RelationalProperty relates at least two Schemas in a formal or physical relationship, while an IntrinsicProperty belongs to exactly one other Schema or Property. This allows to clearly specify the semantics of object-oriented language constructs like classes, associations, association classes and attributes.

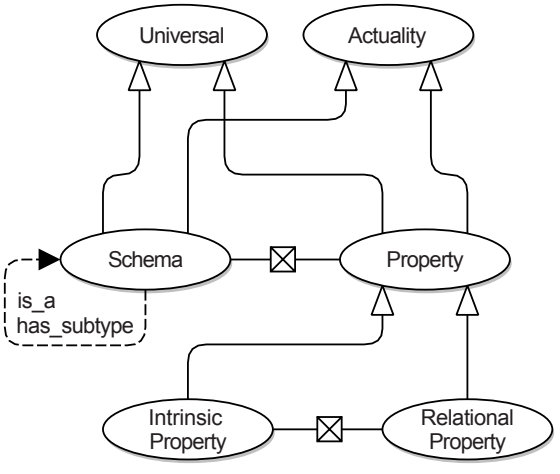

Fig. 3. An overview of universal structural concepts

4.2 Axiomatization

To facilitate inferencing and consistency checks, the semantics of each USMO concept are captured in a rich axiomatization. Since neither description logics nor Horn logic is a subset of the other, we are using a combination of both DL concept constructors and rules for this purpose. Currently, we employ only those constructors available in the OWL standard and emulate features from the recent OWL 1.1 proposal [23] with rules. This applies to axioms such as qualified number restrictions (e.g., $C \sqsubseteq \leqslant np.D$), role inclusion axioms ($R \circ S \dot{\sqsubseteq} R$) and irreflexive, antisymmetric, or disjoint roles.

The entire concept taxonomy of USMO is defined using so-called *covering axioms* [15, p. 75]. This means that for each class C that is extended by a number of subclasses $D_1, ..., D_n$, we add an axiom

$$C \equiv D_1 \sqcup ... \sqcup D_n \tag{1}$$

In addition, we declare all subclasses as disjoint, so an individual instantiating C must necessarily instantiate exactly one of $D_1, ..., D_n$.

Regarding the axiomatization with rules, we employ both *deductive* and *integrity* rules. Deductive rules (also known as derivation rules) assist the modeler by completing the knowledge base with knowledge that logically follows from asserted facts. They are always monotonic. A good example is the rule describing the semantics of the *instantiation* relationship between a Particular p and a Universal u:

$$\mathsf{is\_instance\_of}(p, u) \wedge \mathsf{has\_extension}(u, e) \rightarrow \mathsf{is\_member\_of}(p, e) \tag{2}$$

Integrity rules, by contrast, describe conditions that must hold for the knowledge base to be in a valid state. These rules therefore ensure the *wellformedness* of individual models as well as the *consistency* of all domain-specific viewpoints that constitute a system description.

5 Semantic Web Reasoning over Software Models

As outlined in Sect. 3, we aim at leveraging the power of logical inference and Semantic Web reasoning for integrity and consistency checking of software models. This section describes the practical realization of this goal and discusses solutions to the implementation challenges arising from the open-world assumption in OWL and lack of nonmonotonic queries in SWRL.

5.1 Reasoning Architecture

In Sect. 4.2, we described how the semantics of our new upper ontology are specified using DL axioms and rules. To validate the integrity and consistency of software models using both DL and rule reasoning, we employ a three-layered reasoning architecture (Fig. 4). At the bottom layer, the Jena Semantic Web framework [16] parses OWL ontologies and knowledge bases serialized in RDF. On the middle layer, the Pellet DL reasoner [26] provides basic DL reasoning facilities like subsumption, satisfiability checking, and instance classification. At the top, the rule reasoner of the Jena framework evaluates both deductive rules and nonmonotonic consistency constraints. Each layer provides a view of the RDF graph to the layer above, so the Jena rule engine sees all statements entailed by the Pellet reasoner in addition to those asserted in the knowledge base.

The separation of DL and rule reasoning into two distinct layers is motivated by the following practical considerations: First, ontologies with complex TBox assertions like USMO (many disjunctions, inverse roles, and existential quantifications) require the power of tableau-based implementations available in dedicated DL reasoners such as Pellet. We experienced serious performance degradation when activating the OWL-DL integrity rules in the general-purpose rule engine of the Jena framework. Moreover, Jena rules are not *DL-safe* [21].

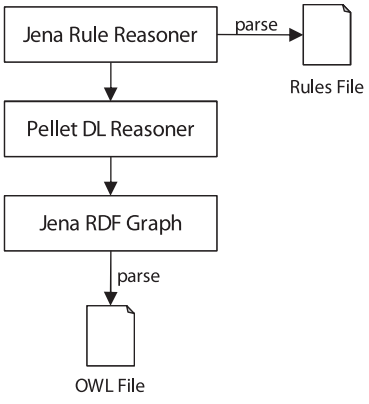

Fig. 4. The Reasoning Architecture in HybridMDSD

When used for DL reasoning, the reasoner creates *blank nodes* for existential quantification and minimum cardinality axioms. As a result, variables in USMO integrity constraints are bound to individuals that are not explicitly asserted in the ABox. This often results in wrongly reported integrity violations.

By contrast, the Pellet reasoner is DL-safe and only returns known individuals from the knowledge base. Unfortunately, Pellet alone does not suffice either. Since it solely processes rules encoded in SWRL, there is no support for nonmonotonic querying over OWL ontologies. In particular, the lack of negation as failure renders it impossible to formulate and validate many USMO integrity constraints. The Jena rule engine does not have these limitations and supports complex nonmonotonic queries via extensible built-ins.

5.2 Simulating a Closed World

One of the major challenges of DL reasoning over software models is the open-world assumption of the Semantic Web. As outlined in Sect. 2, models of software systems usually represent a closed world. Thus, missing model elements might cause integrity constraint violations or inconsistencies between different viewpoints in multi-domain development scenarios. Under the OWA, these violations remain undetected since unknown information is not interpreted as false. Based on the axiomatization of our upper ontology for software models, this section provides a classification of the various problems and discusses possible solutions.

Essentially, we identify three types of USMO axioms negatively affected by the open-world assumption: (1) the covering axioms for "abstract" concepts, (2) existential property restrictions or cardinality axioms and (3) universal quantification axioms.

As an example for covering axioms, consider that every USMO Property is either an IntrinsicProperty or RelationalProperty:

$$\mathsf{Property} \equiv \mathsf{IntrinsicProperty} \sqcup \mathsf{RelationalProperty} \tag{3}$$

Now, an individual p solely asserted to be an instance of Property will *not* violate this constraint, because it trivially satisfies the disjunction [18]; it is just unknown which type of Property p exactly belongs to.

An example for existential quantification is the axiom describing a Property as a dependent entity that cannot exist by itself:

$$\mathsf{Property} \sqsubseteq \exists \mathsf{depends\_on.Schema} \tag{4}$$

Due to the open-world semantics, a Semantic Web reasoner will *not* actually ensure that each Property depends on at least one Schema known in the knowledge base. Similar considerations apply to cardinality restrictions.

Finally, a typical axiom illustrating universal quantification is that Universals cannot depend on Particulars:

$$\mathsf{Universal} \sqsubseteq \forall \mathsf{depends\_on.Universal} \tag{5}$$

Here, a reasoner will not validate the actual type of an individual i asserted as the object in a dependency relationship. Instead, i will be inferred to be of type Universal.

Theoretically, all above-listed axioms can be rewritten with the *epistemic operator* **K** [7] to gain the desired semantics. The **K** operator corresponds to $\Box$ (*Necessarily*) in modal logic and allows to close the world locally for a concept or role [18]:

$$\mathsf{Property} \equiv \mathbf{K}\mathsf{IntrinsicProperty} \sqcup \mathbf{K}\mathsf{RelationalProperty} \tag{6}$$

$$\mathsf{Property} \sqsubseteq \exists \mathbf{K}\mathsf{depends\_on.Schema} \tag{7}$$

$$\mathsf{Universal} \sqsubseteq \forall \mathsf{depends\_on.}\mathbf{K}\mathsf{Universal} \tag{8}$$

Unfortunately, there is currently no support for the **K** operator in OWL, even though a corresponding extension has been suggested many times [18,10,22]. The only implementation known to us is part of the Pellet DL reasoner [6], but it is of prototypical nature and limited to the description logic $\mathcal{ALC}$ [18].

A method to simulate local closed worlds without explicit support for the **K** operator is documented in [22] and [24, pp. 85]. Its main idea is to use set-theoretic operations to determine the set of individuals that *possibly* violate an integrity constraint, in addition to those that are certainly invalid. The key observation is that for a given class C, we can partition all individuals into three distinct sets: (1) those that are known to be a member of C, (2) those that are known to be a member of $\neg C$, and (3) those that may or may not be a member of C. For instance, let C be the class of all individuals that are asserted to be instances of Property, but do not depend on a Schema, thereby causing an integrity violation. In an open world, simply enumerating C does not yield the expected result. The invalid Propertys can instead be found by subtracting everything in group 2 from the set of all Propertys. Obviously, a sufficient condition for individuals in group 2 is that they depend on at least one entity:

$$\mathsf{DependsOnMinOne} \equiv \geqslant 1\ \mathsf{depends\_on} \tag{9}$$

$$\mathsf{InvalidProperty} = \mathsf{Property} - \mathsf{DependsOnMinOne} \tag{10}$$

Unfortunately, there is no DL concept constructor for expressing the difference of classes. Moreover, as observed in [22], this method of querying invalid entities is asymmetric to the usual instance retrieval for other classes. Hence, it does not suit the needs of an end user (i.e., the modeler) who wishes to validate the consistency of the knowledge base in a uniform way.

Since neither the **K** operator nor the set-theoretic approach are feasible in practice, our prototype explicitly "closes the world" before each validation run. In the first case (the covering axioms), this means to explicitly assert for each Property p that it is neither an IntrinsicProperty nor a RelationalProperty if the concrete type is unknown. This results in

$$p \in \mathsf{Property} \sqcap \neg\mathsf{IntrinsicProperty} \sqcap \neg\mathsf{RelationalProperty} \tag{11}$$

and a reasoner will readily report the apparent inconsistency. An alternative approach is to declare the covering subclasses equivalent to an enumeration of all known members. The following listing exemplifies this method in pseudo Java code:

```
1 for each class C {
2   get i_1,...,i_n with {i_1,...,i_n} ⊑ C
3   assert C ≡ {i_1,...,i_n}
4 }
```

In the above example, this results in the following assertions:

$$\mathsf{IntrinsicProperty} \equiv \{\} \tag{12}$$

$$\mathsf{RelationalProperty} \equiv \{\} \tag{13}$$

A reasoner can now detect the inconsistency between an individual p declared to be an instance of Property and the unsatisfiability of the Property concept.

To validate the second type of problematic axioms, namely existential property restrictions or cardinality constraints, it is necessary to close the corresponding roles. This is achieved by asserting for every individual i that a particular role on i will have no more role fillers [3, p. 25]. The following listing illustrates a possible implementation:

```
1 for each individual i {
2   for each class C with i ∈ C {
3     if ( C is one of {∃p.X, ⩾ np, = np} ) {
4       get n with i ∈ = np
5       assert i ∈ ⩽ np
6     }
7   }
8 }
```

For each individual that belongs to a restriction class, we add a membership assertion to an anonymous class that limits the cardinality of the affected

property to the number of current fillers. If this number violates an existential property restriction or cardinality constraint, a reasoner will detect the inconsistency between the different class assertions. Again, the counting of existing property assertions in line 4 is a nonmonotonic operation.

A major disadvantage of this approach is that any subsequent property assertions will render the knowledge base inconsistent. Therefore, if one of the represented models changes, we have to rebuild the entire ABox, which is expensive for large models and makes continuous validation of the knowledge base in the background difficult.

Compared to the problems discussed above, universal quantification axioms are relatively easy to validate. Since our upper ontology has been designed such that subclasses on the same level of the inheritance hierarchy are always pairwise disjoint, a reasoner will detect an inconsistency if fillers for universally quantified properties do not have the expected type.

5.3 Realizing Nonmonotonic Rules

Integrity rules are typically read like "if the body is true, then the head must also be true". Yet, to actually validate these rules and show the results to the user, a logical reformulation is necessary. To see why, consider the following rule that combines the irreflexivity and antisymmetry axioms of the dependency relationship:

$$\mathsf{depends\_on}(x, y) \rightarrow \neg\mathsf{depends\_on}(y, x) \tag{14}$$

Evidently, this rule is nonmonotonic as its head contains a negated atom. In this form, the rule is a statement about the conditions that hold in a consistent closed-world knowledge base **KB**. If we abbreviate the rule with R, we can thus state a new rule S as follows:

$$\neg R \rightarrow \mathit{Inconsistent}(\mathbf{KB}) \tag{15}$$

Actually, we are only interested in those (known) individuals that cause the knowledge base to be inconsistent, so we can reformulate S by inserting the old rule for R and using an auxiliary predicate that classifies an individual as invalid. We can then simply query the knowledge base for all instances of $\mathsf{Invalid}$ and present them to the user:

$$\neg(\mathsf{depends\_on}(x, y) \rightarrow \neg\mathsf{depends\_on}(y, x)) \rightarrow \mathsf{Invalid}(x) \wedge \mathsf{Invalid}(y) \tag{16}$$

Transforming this rule using simple logical equivalences yields:

$$\mathsf{depends\_on}(x, y) \wedge \mathsf{depends\_on}(y, x) \rightarrow \mathsf{Invalid}(x) \wedge \mathsf{Invalid}(y) \tag{17}$$

Obviously, we can rephrase every integrity rule R of the form $B_R \rightarrow H_R$ into a rule S of the form $B_S \rightarrow H_S$, where B_S is $B_R \wedge \neg H_R$. The intuitive reading of S is then "If there are individuals that cause the body of R to be true and the head of R to be false, then these individuals are invalid."

Not all nonmonotonic rules can be as easily transformed as in the example above. Some USMO integrity rules contain negated predicates in their body. We speak of *simple negation* in this case. As an example, consider the following structural integrity rule, which ensures that the containment relationships between entities span a tree rather than a graph:

$$\mathsf{contains}(e_1, e_3) \wedge \mathsf{contains}(e_2, e_3) \wedge e_1 \dot{\neq} e_2 \rightarrow \mathsf{contains}(e_1, e_2) \vee \mathsf{contains}(e_2, e_1) \quad (18)$$

Again, note the nonmonotonicity of this rule caused by the disjunction in the head. Applying the logical transformation outlined above yields:

$$\mathsf{contains}(e_1, e_3) \wedge \mathsf{contains}(e_2, e_3) \wedge e_1 \dot{\neq} e_2 \wedge \neg\mathsf{contains}(e_1, e_2) \wedge \neg\mathsf{contains}(e_2, e_1) \rightarrow \mathsf{Invalid}(e_1) \wedge \mathsf{Invalid}(e_2) \wedge \mathsf{Invalid}(e_3) \quad (19)$$

The disjunction in the head is gone, but we have gained two negated predicates in the body of the new rule. Under an open-world assumption, we do not know for sure whether e_1 contains e_2 or vice versa. Without negation as failure semantics, this rule will never fire.

A more complex problem than simple negation is what we call *negated existential quantification*. Here, the rule body contains an atom that queries the knowledge base for the existence of an individual that matches a number of given criteria. This is required in several USMO integrity rules. As an example, consider the following rule which ensures that the extension of a Universal is not empty:

$$\mathsf{Universal}(u) \wedge \mathsf{has\_extension}(u, e) \rightarrow \exists p\ (\mathsf{Particular}(p) \wedge \mathsf{is\_member\_of}(p, e)) \quad (20)$$

Reformulating this rule according to our transformation guideline yields:

$$\mathsf{Universal}(u) \wedge \mathsf{has\_extension}(u, e) \wedge \neg\exists p\ (\mathsf{Particular}(p) \wedge \mathsf{is\_member\_of}(p, e)) \rightarrow \mathsf{Invalid}(u) \quad (21)$$

Axioms like this one apparently require a nonmonotonic querying mechanism built into the rule language. Some USMO integrity rules even involve existential quantification over not just one, but several variables.

In our implementation, we realize nonmonotonic rules through dedicated built-ins for the Jena rule reasoner. Dealing with simple negation is easy since Jena already offers a nonmonotonic built-in `noValue` that allows to query the knowledge base for concept and role assertions. For instance, Axiom 19 can be expressed as shown in in the following listing:

```
1 [(?e1 usmo:contains ?e3),(?e2 usmo:contains ?e3),notEqual(?e1,?e2),
2     noValue(?e1 usmo:contains ?e2), noValue(?e2 usmo:contains ?e1) ->
3   (?e1 rdf:type sc:Invalid),(?e2 rdf:type sc:Invalid),
4     (?e3 rdf:type sc:Invalid) ]
```

By default, the Jena rule language is not expressive enough to formulate integrity constraints involving negated existential quantification. Fortunately, the set of available built-ins can easily be extended. We have written a custom built-in `notExists` that realizes the required semantics. The following listing exemplifies the usage of this built-in for Axiom 20:

```
[(?u rdf:type usmo:Universal), (?u usmo:has_extension ?e),
    notExists(?p rdf:type usmo:Property, ?p usmo:is_member_of ?e) ->
  (?u rdf:type sc:Invalid) ]
```

6 Related Work

To the best of our knowledge, the HybridMDSD project is the first attempt to leverage Semantic Web reasoning for consistency checking of multiple domain-specific software models based on semantics defined in a shared ontology.

Closely related are the works on UML model consistency validation with description logics by Simmonds et al. [25]. They, too, represent model elements as individuals in the ABox of a DL reasoning system. However, in contrast to our work, they do not employ an axiomatized upper ontology as a semantical foundation and concentrate solely on the UML metamodel, which results in very complex queries and hampers the reusability of their solution.

In the area of multi-domain modeling, Hesselund et al. have presented the *SmartEMF* system [14]. SmartEMF supports validating consistency constraints between several loosely coupled domain-specific languages by mapping all elements of a DSL's metamodel and instantiating models to a Prolog fact base. This yields the advantage of logical queries with closed-world semantics, but it does not offer the powerful features of a DL reasoner such as subsumption and instance classification.

Finally, both the *SemIDE* proposal by Bauer and Roser [2] and the *ModelCVS* project by Kappel et al. [17] aim to integrate different modeling languages based on common domain ontologies. Yet, both approaches focus excusively on an integration on the metamodel level, which essentially means a restriction to TBox reasoning.

7 Conclusion

In this paper, we have analyzed the challenges in Ontology-Driven Software Development that arise from using Semantic Web technologies for representing and reasoning about models in the design and implementation phases. We have particularly focused on the conceptual mismatch between the open-world assumption of the Semantic Web and the closed world of software models in an MDSD process. To exemplify the resulting problems, we have introduced a new upper ontology that allows to semantically integrate different domain-specific software modeling languages. Based on the axiomatization of this ontology, we have illustrated different types of integrity constraints that require closed-world semantics and nonmonotonic reasoning. Our study of critical axioms comprised both DL concept constructors and Horn clause rules. Finally, we described a reasoning architecture that practically realizes the presented techniques.

Acknowledgment. The authors would like to thank Manuel Zabelt and the anonymous reviewers for their valuable comments on previous versions of this

paper. Additionally, we would like to thank explicitly Prof. Uwe Assmann from Chair of Software Technology at Dresden University of Technology, who actively supports the HybridMDSD project.

References

1. Amann, U., Zschaler, S., Wagner, G.: Ontologies, Meta-models, and the Model-Driven Paradigm. In: Ontologies, Meta-models, and the Model-Driven Paradigm, pp. 249–273 (2006)
2. Bauer, B., Roser, S.: Semantic-enabled Software Engineering and Development. In: INFORMATIK 2006 - Informatik für Menschen, Lecture Notes in Informatics (LNI), vol. P-94, pp. 293–296 (2006)
3. Brachman, R.J., McGuiness, D.L., Patel-Schneider, P.F., Resnick, L.A.: Living with CLASSIC: When and How to Use a KL-ONE-Like Language. In: Sowa, J.F. (ed.) Principles of Semantic Networks: Explorations in the representation of knowledge, pp. 401–456. Morgan Kaufmann, San Mateo (1991)
4. Bräuer, M., Lochmann, H.: Towards Semantic Integration of Multiple Domain-Specific Languages Using Ontological Foundations. In: Proceedings of 4th International Workshop on (Software) Language Engineering (ATEM 2007) co-located with MoDELS 2007, Nashville, Tennessee (October 2007)
5. Bruer, M.: Design of a Semantic Connector Model for Composition of Metamodels in the Context of Software Variability. Diplomarbeit, Technische Universität Dresden, Department of Computer Science (November 2007)
6. Clark, L., Parsia: Pellet: The Open Source OWL DL Reasoner (2007), `http://pellet.owldl.com`
7. Donini, F.M., Lenzerini, M., Nardi, D., Nutt, W., Schaerf, A.: An epistemic operator for description logics. Artificial Intelligence 100(1-2), 225–274 (1998)
8. Evermann, J., Wand, Y.: Ontology based object-oriented domain modelling: fundamental concepts. Requirements Engineering 10(2), 146–160 (2005)
9. Gangemi, A., Guarino, N., Masolo, C., Oltramari, A., Schneider, L.: Sweetening Ontologies with DOLCE. In: Gómez-Pérez, A., Benjamins, V.R. (eds.) EKAW 2002. LNCS (LNAI), vol. 2473, Springer, Heidelberg (2002)
10. Grimm, S., Motik, B.: Closed World Reasoning in the Semantic Web through Epistemic Operators. In: OWL: Experiences and Directions (OWLED 2005), Calway, Ireland (November 2005)
11. Guizzardi, G., Herre, H., Wagner, G.: Towards Ontological Foundations for UML Conceptual Models. In: Meersman, R., Tari, Z., et al. (eds.) CoopIS 2002, DOA 2002, and ODBASE 2002. LNCS, vol. 2519, pp. 1100–1117. Springer, Heidelberg (2002)
12. Guizzardi, G., Wagner, G.: Applications of a Unified Foundational Ontology, ch. XIII. In: Some Applications of a Unified Foundational Ontology in Business Modeling, pp. 345–367. IDEA Publisher (2005)
13. Happel, H.-J., Seedorf, S.: Applications of Ontologies in Software Engineering. In: 2nd International Workshop on Semantic Web Enabled Software Engineering (SWESE 2006), held at the 5th International Semantic Web Conference (ISWC 2006), Athens, GA, USA (November 2006)
14. Hessellund, A., Czarnecki, K., Wasowski, A.: Guided Development with Multiple Domain-Specific Languages. In: Engels, G., Opdyke, B., Schmidt, D.C., Weil, F. (eds.) MODELS 2007. LNCS, vol. 4735, Springer, Heidelberg (2007)

15. Horridge, M., Knublauch, H., Rector, A., Stevens, R., Wroe, C.: A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools Edition 1.0. The University Of Manchester (August 2004)
16. Jena – A Semantic Web Framework for Java. Version 2.5 (January 2007), http://jena.sourceforge.net
17. Kappel, G., Kramler, G., Kapsammer, E., Reiter, T., Retschitzegger, W., Schwinger, W.: ModelCVS - A Semantic Infrastructure for Model-based Tool Integration. Technical Report, Business Informatics Group, Wien Technical University (July 2005)
18. Katz, Y., Parsia, B.: Towards a Nonmonotonic Extension to OWL. In: OWL: Experiences and Directions (OWLED 2005), Calway, Ireland (November 2005)
19. Kühne, T.: Matters of (Meta-) Modeling. Software and Systems Modeling 5(4), 369–385 (2006)
20. Lochmann, H.: Towards Connecting Application Parts for Reduced Effort in Feature Implementations. In: Proceedings of 2nd IFIP Central and East European Conference on Software Engineering Techniques CEE-SET 2007 (WIP Track), Posen, Poland (October 2007)
21. Motik, B., Sattler, U., Studer, R.: Query Answering for OWL-DL with Rules. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, Springer, Heidelberg (2004)
22. Ng, G.: Open vs Closed world, Rules vs Queries: Use cases from Industry. In: OWL: Experiences and Directions (OWLED 2005), Calway, Ireland (November 2005)
23. Patel-Schneider, P.F., Horrocks, I.: OWL 1.1 Web Ontology Language Overview. World Wide Web Consortium (W3C), W3C Member Submission (December 2006), http://www.w3.org/Submission/owl11-overview/
24. Racer Systems GmbH & Co. KG. RacerPro User's Guide Version 1.9 (December 2005)
25. Simmonds, J., Bastarrica, M.C.: A tool for automatic UML model consistency checking. In: Proceedings of the 20th IEEE/ACM international Conference on Automated Software Engineering (ASE 2005), Long Beach, CA, USA, Demonstration Session: Formal tool demo presentations, pp. 431–432. ACM Press, New York (2005)
26. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A Practical OWL-DL Reasoner. Journal of Web Semantics (submitted for publication, 2007)
27. Stahl, T., Völter, M.: Model-Driven Software Development: Technology, Engineering, Management, 1st edn. Wiley & Sons, Chichester (2006)