

# Towards Semantic Integration of Multiple Domain-Specific Languages Using Ontological Foundations\*

Matthias Bräuer and Henrik Lochmann

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

**Abstract.** Model-Driven Software Development (MDS) advocates the use of domain-specific modeling languages (DSLs) for describing software systems. Modeling complex systems usually requires several different DSLs, which raises the need to consistently integrate the corresponding models. However, current model-driven approaches do not properly support expressing semantic relationships and interdependencies between distinct modeling languages. Often, this information is implicitly hidden in code generators, weaving models, or model transformations, which hampers the reusability of DSLs and associated assets. We present an approach that facilitates the integration of domain-specific languages on a semantic level by mapping language constructs to concepts in an upper ontology. In contrast to previous approaches, we concentrate on the system design phase. We present a customized upper ontology for software modeling and demonstrate the feasibility of our approach in a case study.

## 1 Introduction

In recent years, the importance of domain-specific languages (DSLs) for describing software systems has increased and a convergence with model-driven software development (MDS) can be witnessed [5]. Compared with large general-purpose modeling languages such as the UML, DSLs only offer a limited set of constructs. This increases modeling productivity and facilitates a precise definition of concerns within a particular domain [32]. However, modeling complex systems usually necessitates the use of multiple DSLs [12, 19] resulting in several domain-specific models, each describing one aspect of an application (e.g., data structures, workflows, or user interfaces). It is believed that many small, loosely-coupled models are easier to handle and improve readability in comparison with a single, monolithic model [36]. Yet, *multi-domain modeling* also raises the need for sophisticated consistency checking between the different viewpoints. This is particularly vital when models frequently change, e.g., in iterative development or as a result of feature configurations in a software product line [7].

Current model-driven approaches typically express the interdependencies between elements in related models via name-based references, weaving models, or mapping

---

\* This work is supported by the feasiPLE project financed by the German Ministry of Education and Research (BMBF).

specifications (e.g., graph rewrite rules or model transformations). Unfortunately, these methods do not specify the *semantic relationships* between individual model elements explicitly. Information about the meaning of a model element in relation to another one is hidden in weaving links or model transformation rules. Moreover, the pairwise connection of models does not scale for complex system descriptions with many DSLs, and it hampers the reuse of modeling languages in different language combinations.

In an ongoing research project, we are currently investigating how semantic technologies, in particular ontologies and the Semantic Web standards, can help to alleviate these problems. Our project is in line with other recent efforts that propagate the combined use of both the MDSD and the Ontology technical space [11, 17]. Our idea is to use an ontology knowledge base as a *semantic connector* for multiple domain-specific models. By reusing concepts and relationships from suitable domain, core, and upper ontologies, we hope to provide a semantically rich system description that can be leveraged for automatic reasoning and consistency checking. In this paper, we report on our initial findings, namely the definition of a common upper ontology for software modeling languages and its successful application in a simple case study that involves the integration of different DSLs.

The remainder of the paper is structured as follows: After describing the problem and motivation for our project in Sect. 2, we present the core ideas and expected benefits of our approach in Sect. 3. Section 4 discusses ontological foundations for modeling languages and contains our proposal for an upper ontology of software design modeling languages. Section 5 illustrates the application of our approach in a small case study. In Sect. 6, we briefly highlight related work to place our own approach in context. Finally, Sect. 7 concludes the paper and gives an outlook on future work.

## 2 Problem and Motivation

In MDSD, domain-specific modeling languages are usually specified using *metamodeling* [32]. A metamodel defines the concepts of a language and their relationships (viz., the *abstract syntax*) as well as a set of wellformedness rules (viz., the *static semantics*) which govern the validity of models written in that language. In multi-DSL development scenarios, this does not suffice to ensure the consistency and validity of all models constituting a complex system description, because the semantic relationships between constructs from different languages remain unspecified. In the literature, several solutions to the problem of multi-domain integration have been proposed:

**Name-based references.** Warmer and Kleppe [36] suggest to connect models by referencing model elements by name from within other models. This solution has the drawback that it delegates all information about the semantic interrelationship and dynamic interplay of different language constructs to the code generator. In other words, the actual integration happens at the code level; the references on the model level do not reveal how two model elements are semantically related.

**Weaving models.** A model weaver allows to define and visualize correspondences between elements in different models using an extensible weaving metamodel [9]. The advantage of this approach is that references between model elements are kept

outside of the models. However, a weaving model can only express the semantics of the *links* between different model elements (e.g., denoting their equivalence); it does not actually define the *meaning* of the corresponding language constructs. Also, a model weaver usually connects pairs of models and thus is not suited for an integration of a large number of domain-specific languages.

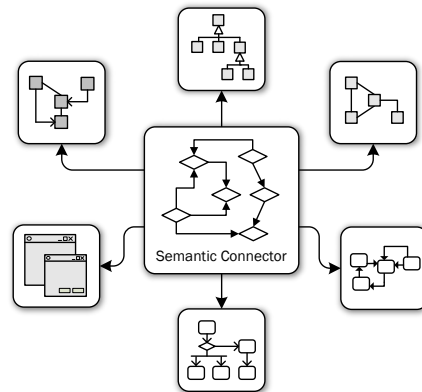
**Model mappings.** Mappings between models are the cornerstone of many MDSD approaches such as the OMG MDA initiative [27]. They can be realized as XSLT transformations, graph rewrite rules (e.g., triple graph grammars), or metamodel-based model transformations. Given a suitable formalism, these mappings can be seen as models themselves [4], which describe the relationships between elements from the source and target model and allow to validate inter-model consistency constraints. However, most transformations between models are written in an ad-hoc fashion and are specific to the respective source and target metamodel. This severely limits their reuse potential and hampers an easy integration of arbitrary domain-specific languages. Furthermore, a mapping specification expresses semantics of different DSL elements only implicitly and does not assign an explicit meaning to each language construct.

To address the insufficiencies of current approaches, a number of recent proposals argue in favor of *ontologies* as a means to specify language semantics and integrate multiple domain-specific modeling languages [3, 21]. We agree with these views, but additionally stipulate that language integration based on ontological foundations must be guided by a common upper ontology for software modeling languages. This allows to reuse fundamental relationships, constraints, and axioms when integrating a new DSL. Defining such an upper ontology is the topic of this paper.

### 3 Integrating Languages with Ontologies

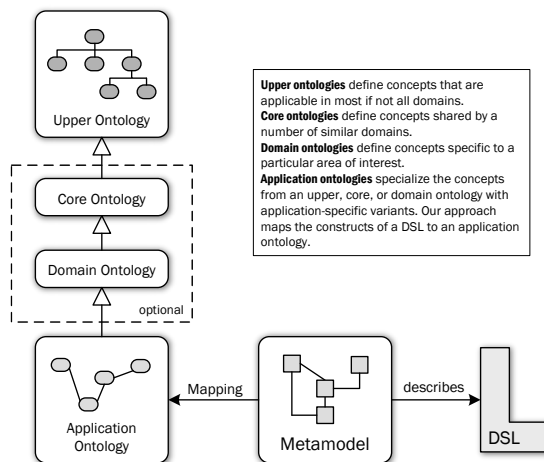
The core idea of our approach, which we have coined *semantic model connector*, is to establish semantic links between different domain-specific models using an ontology knowledge base. In contrast to a weaving model, the connector does not store information about *correspondences* between model elements, but contains *individuals* (i.e., instances of ontology classes) and property assertions that *represent* one or several model elements and their relationships. Note that we do not intend to create representatives for all model elements. One central aspect of our methodology is to restrict the connector knowledge base to those elements required for a semantic connection between the participating models. Thus, the connector forms a less detailed, but holistic view of the entire system being modeled. Figure 1 illustrates this central concept formation.

To automatically populate the connector knowledge base with representatives for newly created model elements, we need to map the relevant meta-classes and meta-properties of each DSL's metamodel to the concepts and relationships in a common reference ontology. Essentially, this mapping defines the semantics of modeling language constructs based on the concept descriptions and axioms in the ontology. Of course, an ontology mapping lacks the rigor of a complete operational semantics specification, but in many cases it will suffice to express cross-language relationships and



**Fig. 1.** Integrating different domain-specific models with a semantic connector

constraints. In the current phase of the project, we employ an *upper ontology* with very general concepts and only a limited set of relationships, but we realize that more specialized ontologies will eventually be necessary to define detailed DSL mappings (cf. Figure 2). In the long term, we envisage a library of reusable DSLs managed within a project team or company, where each language comes with a semantic binding to the common foundational ontology or a shared set of core and domain ontologies.



**Fig. 2.** A hierarchy of ontologies for defining language semantics

The motivation for starting with an upper ontology is to provide a number of well-defined, theoretically principled, and sufficiently axiomatized concepts as a base for concepts in more specialized ontologies. As argued in [26], the resulting core and domain ontologies usually have a higher quality than those developed bottom-up from a selection of sample languages. We will describe our experimental upper ontology and its creation in more detail in Sect. 4.

Obviously, the constructs of a domain-specific language will not always map one-to-one to the concepts in the reference ontology. One of our research goals therefore is to investigate how language designers can be assisted in writing complex mapping rules between the two technical spaces. Graph-rewrite rules based on visual patterns are a promising solution to this problem. Fortunately, the mapping to a central ontology has only linear complexity ( $O(n)$ ) for a growing number  $n$  of DSLs, compared with  $O(n^2)$  for pairwise model transformations between each language.

Now, a likely objection to our approach is that a so-called “pivot model”, i.e., an intermediate and universal metamodel, provides similar possibilities for integrating the metamodels of multiple DSLs, using a constraint language such as OCL for defining inter-model consistency rules. As a general answer, we point to the underlying presumption of our project that combining MDS and ontologies leverages the respective strengths of each technical space [33], but we also envisage a number of concrete benefits:

**Lightweight coupling.** Semantic Web technologies provide the necessary means for effective decoupling of different languages and models by storing references solely as textual URIs. By design, these are independent of particular language specification technologies such as MOF [28] or EMF [6]. A pivot model is necessarily biased towards the platform it has been defined for.

**Extensibility.** The Web Ontology Language (OWL) [31] permits to flexibly derive subclasses from concepts in an ontology without requiring access to the original file. In OWL, information can be added to anything anywhere. This facilitates an easy creation of DSL-specific application ontologies. Moreover, ontology classes can be defined based on value or cardinality restrictions, which allows to specify consistency rules more intuitively than with a metamodeling-based constraint language.

**Reasoning support.** Due to their formal logical underpinning, Semantic Web technologies offer powerful means for reasoning and inferencing. Since the semantic connector contains individuals, we are not limited to subsumption over concept hierarchies, but can take advantage of full instance-level reasoning in a knowledge base. For example, a reasoner can entail and validate additional semantic relationships and automatically insert or delete elements to keep the knowledge base consistent. Experiments with representing models as a Prolog fact base [19] have shown that conventional model querying languages do not match the power of logics-based approaches.

## 4 Deriving an Upper Ontology for Software Modeling Languages

A first step towards realizing our project goals was to investigate which primitive ontological categories are required to describe the semantics of software models. Initially,

we aimed at integrating languages from all abstraction levels. Interestingly, there is a rich body of previous research on ontological foundations for conceptual modeling languages. However, to the best of our knowledge, there is no published work that deals with upper ontologies for modeling languages used in the system design or implementation phases. In the following, we mention existing proposals from the area of conceptual modeling. We briefly assess their applicability and discuss the problem of formalization. Finally, we describe an experimental upper ontology for software modeling languages which is used in the prototypical implementation of the semantic connector.

#### 4.1 Ontological foundations for conceptual modeling languages

*Conceptual modeling* is among the most important activities in the initial phases of the software development process (i.e., requirements engineering and system analysis). It aims at describing the business domain (viz., the “real world”) in abstract terms and without the technical bias that characterizes models in the subsequent design phase.

In the past, several proposals have tried to provide sound semantics for conceptual UML models by mapping UML constructs to the concepts of an upper ontology. In their pioneering work [8], Evermann and Wand employed the *Bunge-Wand-Weber (BWW)* ontology [35] — arguably the most prominent approach to ontology-based information systems modeling — for this purpose. In a related study, Opdahl and Henderson-Sellers also used BWW for an ontological evaluation of the UML [29]. In great similarity, Guizzardi et al. [15] defined ontological foundations for conceptual UML models using the *General Ontological Language (GOL)* [18] and its upper ontology called *GFO*. This led to the development of the *Unified Foundational Ontology (UFO)* and a comprehensive theory of structural conceptual models [13].

Naturally, the question arises whether these results can be translated to the problem of defining the semantics of domain-specific modeling languages used in system design. Indeed, Evermann and Wand note that the “derived rules might not be applicable for [...] software modelling”. We have analyzed different structural conceptual models that were created using the design patterns suggested by Guizzardi et al. We found that in some cases the ontological metatype of model elements changed when moving from analysis to design. A complete presentation of these findings is outside the scope of this paper, so we solely state our conclusion that reference ontologies for conceptual modeling languages cannot directly be used for expressing the semantics of design modeling languages. Yet, we believe that some general concepts found in these ontologies can guide the development of a dedicated upper ontology for software modeling languages (cf. Sect. 4.3).

#### 4.2 The Question of Formalization

Guizzardi et al. have argued that the design of a domain-specific visual language (DSVL) should be based on a formal domain ontology [16]. More precisely, they have established a number of properties that need to hold to guarantee an isomorphic mapping between the metamodel of a DSVL and an ontology representing the associated domain. They postulate that the domain ontology’s full axiomatization must be transferred to the metamodel of the DSVL and that this ontology needs to be formulated in a general

ontology representation language based on a philosophically principled foundational ontology.

We consider these requirements to be difficult to fulfill in a practical setting. Firstly, domain-specific languages in an MDSO process constitute development assets that many other artifacts depend on. The redesign of existing DSLs may thus have repercussions on models, code generators, constraint definitions, etc. Secondly, it is questionable whether DSL designers without a philosophical background will tailor their languages to fit into the rigid axiomatization prescribed by a foundational ontology. As Kurtev has observed [23], many existing languages “violate the ‘ideal’ ontological commitment”. Due to time and budget constraints, a formal top-down approach to language design is usually not feasible in practice. Indeed, many proponents of model-driven development approaches advocate an iterative process incorporating new language elements over time and adapt semantics accordingly [32, p. 113]. Obviously, this does not harmonize well with the ontological soundness demanded by Guizzardi et al. upfront. Finally, it is unlikely that the complex axiomatization of a domain or foundational ontology easily translates to a constraint language in the MDSO world.

As an answer, we propose to define the semantics of the upper ontology’s concepts in a less rigid way using the ontology and rule languages developed by the Semantic Web community. Within the limited scope of the DSL library of a development team, such a “light-weight” approach will usually not exhibit the information integration and semantic interoperability problems reported by Guizzardi in [14]. In essence, we use upper ontologies for terminological services and inferencing based on a small set of concepts whose semantics are known, rather than for meaning negotiation among heterogeneous agents as suggested in [26].

### 4.3 An Experimental Upper Ontology

As explained further above, we aim for an upper ontology that provides the basic concepts for domain-specific modeling languages used in system design and implementation. We currently cannot provide a solution for the (desirable) goal of integrating conceptual analysis models due to the differences in their ontological foundations (cf. Section 4.1). Since our project goals are practicality and applicability in industrial use cases, we need to define a set of foundational concepts that is small and concise enough to remain manageable in practical scenarios, but has enough ontological power to cover a wide range of different DSLs.

To meet these requirements, we have first developed a framework for classifying domain-specific system modeling languages. The three dimensions of this framework are:

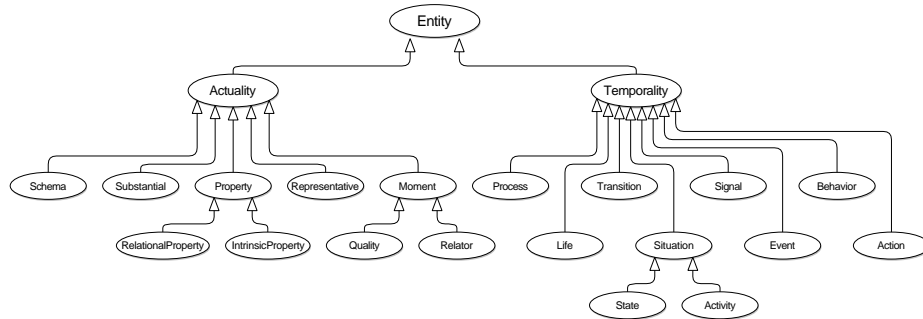
**Solution domain.** Refers to the *horizontal domain* [7] addressed by a DSL and can be broadly categorized into *structural*, *behavioral* and *UI* languages.

**Abstraction level.** Follows the suggestion of the OMG MDA framework [27] and characterizes the constructs of a modeling language as *computation-independent*, *platform-independent*, or *platform-specific*.

**Ontological kind of models.** Partitions modeling languages according to the nature of elements in resulting models. This distinction is based on works by Kühne [22] and

determines whether model elements abstract from a modeled system by *classifying* the individual runtime objects into types (*type models*) or represent them with a one-to-one image (*token models*). Typical examples for type and token models are UML class and object diagrams, respectively.

We used the framework to categorize a large number of sample languages to gain a feeling for recurring concepts and DSL design patterns. We proceeded with adapting and extending a small, lightweight upper ontology called ABC [20], which features an intuitive design and already contains many key concepts of more powerful ontologies. We carefully analyzed shortcomings of the ABC ontology with respect to the requirements listed above and incorporated elements from BWW, GOL/GFO, UFO, and the DOLCE ontology of particulars [10]. Subsequently, we identified further missing elements by mapping language constructs from our set of sample languages to the ontology. This way, we iteratively implemented enhancements for the basic conceptual structures. An excerpt of the resulting upper ontology is shown in Fig. 3.



**Fig. 3.** An upper ontology for software modeling languages

In line with other philosophically-motivated approaches, we draw a major distinction between enduring and perduring entities. Enduring entities (or *Actualities*) are “wholly present [...] at any time” [10], while perduring entities (or *Temporalities*) are only partially present in time. We can say that enduring entities *are in time* while perduring entities *happen in time*. In addition to these fundamental conceptual categories, we distinguish between universal entities (or *universals*) and particular entities (or *particulars*). This distinction is very important to integrate both modeling languages for type models and those for token models. As an example, consider the concepts *Schema* and *Substantial*, which represent universal and particular data structures, respectively. The two concepts are related via an *instance\_of* property. This relationship also exists between the concepts *Property* and *Moment* (including respective subclasses), *Signal* and *Event* as well as *Behavior* and *Action*.

## 5 Case Study

In the following, we will illustrate the application of our approach in a simple case study. To this end, we have modeled a conference registration system using three different domain-specific languages. Our system description comprises a *structural model* of the business entities, a *dialog model* describing the page flow through the application and a *presentation model* of the user interface for each screen.<sup>1</sup> For brevity reasons, we will not discuss the entire mapping to the connector but concentrate on a few selected issues that showcase the potential.

**Connecting the Structural Model with the Presentation Model** As a first example, consider the well-known problem of binding user interface components to data structures. Figure 4 illustrates how the semantic connector can achieve consistency between the presentation model of the UI and the structural model describing the data. The upper part shows an excerpt of the corresponding metamodels, while the lower part highlights the semantic mapping provided by the corresponding DSL designers. For each relevant language construct, we derive a subclass from the corresponding upper ontology concept. At the moment, we employ the Eclipse Modeling Framework (EMF) [6] for metamodeling the DSLs, so we can easily establish a reference between the two technical spaces via the URI of each metamodel element.

This mapping exhibits a number of interesting properties. First, we want to emphasize that models written in the structure DSL are type models, while the presentation DSL results in token models (cf. Sect. 4.3). From an ontological point of view, the UI elements in the presentation language are particulars. In this case, we map them to the `Representative` concept to indicate that they do not themselves hold data but merely *represent* it in the UI. Representatives are very flexible and may stand for all types of particulars, even processes and events. Now, to semantically link the two languages, our upper ontology introduces the *modal* property `may_represent` between the `Representative` and the `Universal` concepts. Since we support both universals and particulars, concrete instances of a `BusinessObject` (e.g., in a data flow diagram) can be modeled as well by extending the `Substantial` concept. Figure 4 explains how this fits into the general ontological structure. We are currently experimenting with inference rules (expressed in the rule language of the Jena Semantic Web framework [2]) to check the consistency of the knowledge base when both universals and particulars appear in it.

Furthermore, property restrictions in OWL enable the DSL designer to restrict the range of the `may_represent` property. Therefore, the ontological counterpart of a `Textfield` instance may only represent `DataField` instances. Further ontology subclasses allow more fine-grained control to distinguish, e.g., between different types of textfields in the connector. Additionally, the reasoning and querying engine can suggest to the UI modeler which datafields can be assigned to a textfield in a particular context. We plan to integrate this feature as a form of “model autocompletion” hint available in our envisaged tool suite.

---

<sup>1</sup> The example has been inspired by an application mentioned in [25] which, however, used only a single DSL.

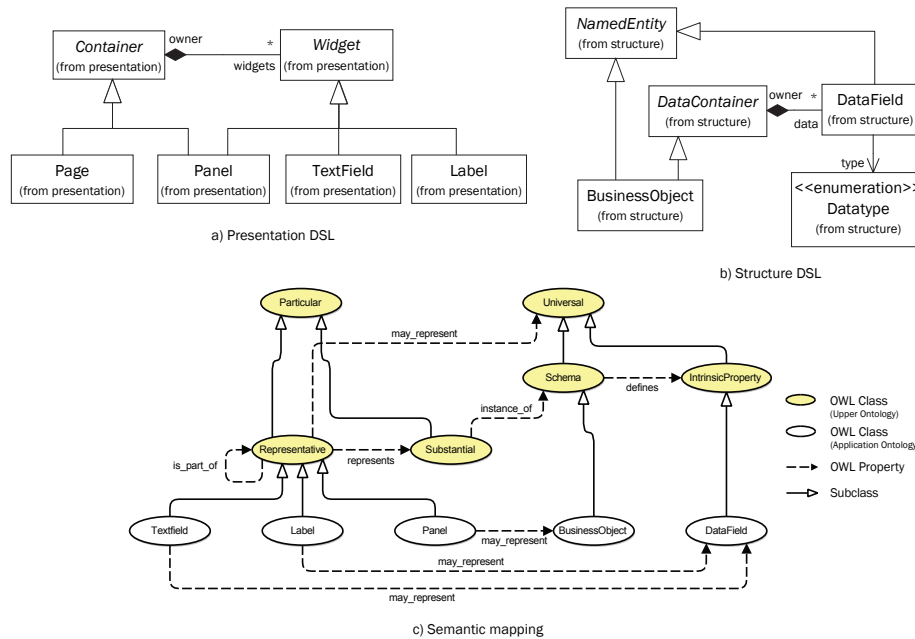


Fig. 4. Semantic mapping of the structure and the presentation DSL

**Connecting the presentation language with the dialog flow language** This time, we will have a look at the instantiated connector. Figure 5 depicts how the UI presentation model of a page is semantically integrated with the dialog model that describes the page flow. In the presentation model, the page is mapped to an instance of a subclass of Representative. In the dialog model, a page node represents an Activity. For enhanced clarity, we have added the names of the corresponding upper ontology concepts in the diagram. Notice that the semantic structures are created automatically while domain experts build the associated models. This logic is based on the semantic mapping defined by the DSL designers.

As pointed out above, inference rules that define axioms for the upper ontology may help to populate the connector knowledge base with further entailed facts thereby alleviating the need for highly complex mapping expressions.

To get a glimpse of the potential benefits of our approach, assume the existence of the following general consistency rule:

$$\begin{aligned}
 & \text{depends\_on}(p1, p2) \wedge \text{represents}(r1, p1) \wedge \\
 & \text{represents}(r2, p2) \rightarrow \text{depends\_on}(r1, r2)
 \end{aligned}$$

For better legibility, we have omitted the `rdf:type` statements. In plain English, the rule states that a dependency relationship between two particulars ( $p1, p2$ ) implies a dependency between the corresponding representatives ( $r1, r2$ ). Now, consider that both

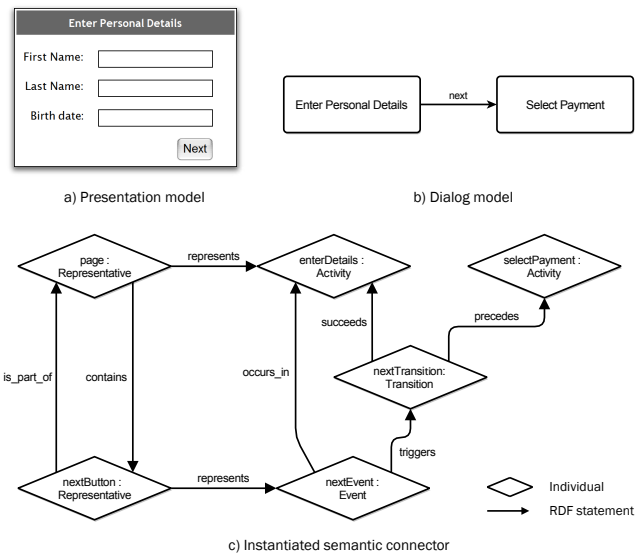


Fig. 5. Semantic Connector of a presentation and a dialog model

the `is_part_of` and the `occurs_in` property are subproperties of `depends_on`. A Semantic Web reasoner can use this information and infer that the rule holds for the model depicted in Fig. 5. However, if we moved the “Next” button to another page, the reasoner would flag an inter-model inconsistency. Similar rules could ensure that there is a corresponding event for each button added to the UI model. Ultimately, our goal is to leverage the semantic connector for automatic or semi-automatic adaptation of related models in the face of changes to one model. For a large number of different languages, we expect efficiency gains compared to a “traditional” approach based on model transformations.

## 6 Related Work

Semantic integration of modeling languages using ontological foundations is a relatively new field of research. Although potential benefits of this approach have been identified as early as 1996 [34], concrete proposals have only started to appear during the last years.

In [3], Roser and Bauer sketch the idea of a *Semantic IDE* which maintains a representation of a DSL both as a metamodel and as an application ontology. The authors suggest to bind all application ontologies to a reference ontology to “obtain unambiguous semantics” and “facilitate more sophisticated reasoning”. Yet, they do not further elaborate on the necessary expressiveness of such a reference ontology and simply assume its general availability. In a related article [30], they employ OWL-S [24] as an ontological reference to automatically derive a model transformation between the meta-

models of two process modeling languages. However, they do not consider the semantic integration of DSLs covering very heterogeneous domains.

In great similarity, Kramler et al. [21] propose a semantic infrastructure called *ModelCVS* that allows to integrate models from different tools and languages. They, too, create application ontologies by mapping elements in the languages' metamodels to corresponding concepts in the ontology. The authors briefly mention the *alignment* of models written in conceptually distinct languages, but do not explicitly address multi-domain modeling. Their case study deals with *translating* UML activity diagrams into BPEL models and vice versa using a generic workflow ontology. Also, they do not connect models using an ontology knowledge base, but derive bridging operators and model transformations solely on the metamodel layer. As a result, their use of an ontology reasoner is restricted to the discovery of subsumption relationships between concepts in a taxonomy. Lastly, the ModelCVS system assumes a complete "lifting" of a DSL metamodel into a corresponding ontology. In contrast, we transfer only semantically relevant metamodel classes into the Ontology technical space.

## 7 Conclusion and Outlook

In this paper, we approached the challenge of software systems engineering with multiple models written in different domain-specific languages. In line with other authors, we proposed the use of ontologies to describe the semantic relationships and interdependencies between these languages. We presented our own approach which is grounded on the idea of expressing semantic links between multiple domain-specific models using an ontology knowledge base. We discussed an experimental upper ontology for software modeling languages that aims at providing the key concepts to integrate a wide variety of DSLs. A case study provided first insights into the applicability and usefulness of our method. Finally, we gave an overview over related studies and highlighted differences to our approach.

At the moment, we are working on the implementation of the methodology introduced in this paper. We envisage an Eclipse-based tool suite with visualization and editing facilities built around the semantic connector knowledge base. Desirable features include automatic consistency checking, navigation along semantic links, model "autocompletion" based on instance-level reasoning and querying (cf. Sect. 3), visualization of semantic relationships, pattern-oriented rule editing, generation of system architecture documentation, etc. The benefit of these features is twofold: on the one hand, multiple models in a multi-domain development scenario are kept consistent, which is particularly vital to check the validity of feature configurations in a software product line. On the other hand, domain experts can better understand how their modeling artifacts fit into the overall system description. By making interdependencies between models explicit, our approach fosters traceability and improves communication between the model developers. Moreover, we expect the holistic, semantics-based system description via the connector to aid architects, product management, and other stakeholders wishing to see the big picture of an application rather than all the fine details.

We cannot yet provide a detailed evaluation of the presented ontology and the applicability of the entire approach in a practical project. For this purpose, we are currently

realizing a large-scale case study within the research project feasiPLe [1]. We hope to deepen our understanding of the ontological foundations of system modeling languages and identify additional requirements for enhancements of our approach. Within the case study, we develop a customer-related product management system that requires the integration of up to five different domain-specific modeling languages.

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

## References

1. feasiPLe - Feature-driven, aspect-oriented and model-driven Software Product Line Development. [http://www.feasiple.de/index\\_en.html](http://www.feasiple.de/index_en.html), 2006.
2. Jena – A Semantic Web Framework for Java. <http://jena.sourceforge.net>, Jan. 2007. Version 2.5.
3. B. Bauer and S. Roser. Semantic-enabled Software Engineering and Development. *INFORMATIK 2006 - Informatik für Menschen, Lecture Notes in Informatics (LNI)*, P-94:293–296, 2006.
4. J. Bézivin, F. Büttner, M. Gogolla, F. Jouault, I. Kurtev, and A. Lindow. Model Transformations? Transformation Models! In O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, editors, *Proceedings 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS'2006)*, volume 4199 of *LNCS*, Berlin, 2006. Springer.
5. J. Bézivin, F. Jouault, I. Kurtev, and P. Valduriez. Model-Based DSL Frameworks. In *Companion to the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2006*, pages 602–616, Portland, Oregon, USA, Oct. 2006. ACM Press.
6. F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, and T. J. Grose. *Eclipse Modeling Framework*. Eclipse Series. Addison Wesley Longman, Amsterdam, 1st edition, Aug. 2003.
7. K. Czarnecki and U. W. Eisenecker. *Generative Programming. Methods, Tools and Applications*. Addison-Wesley Longman, Amsterdam, 2000.
8. J. Evermann and Y. Wand. Towards Ontologically Based Semantics for UML Constructs. In H. S. Kunii, S. Jajodia, and A. Sølvberg, editors, *Proceedings ER 2001. 20th International Conference on Conceptual Modeling, Yokohama, Japan*, volume 2224 of *LNCS*, pages 354–367, Berlin Heidelberg, 2001. Springer.
9. M. D. D. Fabro, J. Bézivin, F. Jouault, E. Breton, and G. Gueltas. AMW: A Generic Model Weaver. In *Journées sur l'Ingénierie Dirigée par les Modèles (IDM05)*, Paris, France, June 2005.
10. A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, and L. Schneider. Sweetening Ontologies with DOLCE. In *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW'02)*, Madrid, Spain, Oct. 2002. Springer.
11. D. Gašević, D. Djuric, and V. Devedžic. Bridging MDA and OWL Ontologies. *Journal of Web Engineering*, 4(2), 2005.
12. J. Greenfield and K. Short. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. Wiley & Sons, Indianapolis, Indiana, USA, 2004.
13. G. Guizzardi. *Ontological Foundations for Structural Conceptual Models*. PhD thesis, University of Twente, Enschede, The Netherlands, 2005. Telematica Instituut Fundamental Research Series No. 15.

14. G. Guizzardi. The Role of Foundational Ontologies for Conceptual Modeling and Domain Ontology Representation. In *7th International Baltic Conference on Databases and Information Systems, Vilnius, Lithuania, 2006*. Companion Paper for the Invited Keynote Speech.
15. G. Guizzardi, H. Herre, and G. Wagner. Towards Ontological Foundations for UML Conceptual Models. In R. Meersmann and Z. Tari, editors, *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, ODBASE. Proceedings of the Confederated International Conferences, Irvine, California, volume 2519 of LNCS*, pages 1100–1117, Berlin, 2002. Springer.
16. G. Guizzardi, L. F. Pires, and M. van Sinderen. An Ontology-Based Approach for Evaluating the Domain Appropriateness and Comprehensibility Appropriateness of Modeling Languages. In *Proceedings of the 8th MoDELS, Jamaica*, pages 691–705, 2005.
17. H.-J. Happel and S. Seedorf. 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, Nov. 2006.
18. B. Heller and H. Herre. Ontological Categories in GOL. In J. Seibt, editor, *Process Theories: Crossdisciplinary Studies in Dynamic Categories*, pages 57–77. Kluwer Academic Publishers, Dordrecht, 2003.
19. A. Hessellund, K. Czarniecki, and A. Wasowski. Guided Development with Multiple Domain-Specific Languages. In *ACM/IEEE 10th International Conference On Model Driven Engineering Languages and Systems (MODELS 2007), Nashville, TN, USA, Sept./Oct. 2007*. To appear.
20. J. Hunter. Enhancing the Semantic Interoperability of Multimedia through a Core Ontology. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(1):49–58, 2003.
21. G. Kramler, G. Kappel, T. Reiter, E. Kapsammer, W. Retschitzegger, and W. Schwinger. Towards a Semantic Infrastructure Supporting Model-based Tool Integration. In *International Conference on Software Engineering, Proceedings of the 2006 International Workshop on Global Integrated Model Management, Shanghai, China, Session Metamodels and Semantics*, pages 43–46, New York, NY, USA, 2006. ACM Press.
22. T. Kühne. Matters of (Meta-) Modeling. *Software and Systems Modeling*, 5(4):369–385, Dec. 2006.
23. I. Kurtev. Metamodels: Definitions of Structures or Ontological Commitments? In *Workshop on TOWERS of models. Collocated with TOOLS Europe 2007*, 2007.
24. D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan, and K. Sycara. Bringing Semantics to Web Services: The OWL-S Approach. In J. Cardoso and A. Sheth, editors, *Proceedings of First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004), July 2004, San Diego, California, USA*, volume 3387 of LNCS, pages 26–42, Berlin Heidelberg, July 2005. Springer.
25. MetaCase. Domain-Specific Modeling with MetaEdit+: 10 Times Faster Than UML. White Paper, MetaCase Consulting, 2007. [http://www.metacase.com/papers/Domain-specific\\_modeling\\_10X\\_faster\\_than\\_UML.pdf](http://www.metacase.com/papers/Domain-specific_modeling_10X_faster_than_UML.pdf).
26. D. Oberle. *Semantic Management of Middleware*, volume 1 of *Semantic Web and Beyond: Computing for Human Experience*. Springer, Berlin, 2006.
27. Object Management Group (OMG). *MDA Guide Version 1.0.1*, June 2003. OMG document omg/2003-06-01, <http://www.omg.org/docs/omg/03-06-01.pdf>.
28. Object Management Group (OMG). *Meta Object Facility (MOF) Core Specification, Version 2.0*, Jan. 2006. OMG document formal/05-07-04, <http://www.omg.org/docs/formal/05-07-04.pdf>.
29. A. L. Opdahl and B. Henderson-Sellers. Ontological Evaluation of the UML Using the Bunge–Wand–Weber Model. *Software and Systems Modeling*, 1(1):43–67, Sept. 2002.

30. S. Roser and B. Bauer. An Approach to Automatically Generated Model Transformations Using Ontology Engineering Space. 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, Nov. 2006.
31. M. K. Smith, C. Welty, and D. L. McGuinness. *OWL Web Ontology Language Guide*. World Wide Web Consortium (W3C), Feb. 2004. W3C Recommendation, Available at: <http://www.w3.org/TR/owl-guide/>.
32. T. Stahl, M. Völter, S. Efftinge, and A. Haase. *Modellgetriebene Softwareentwicklung: Techniken, Engineering, Management*. dpunkt.verlag, Heidelberg, 2nd edition, 2007. In German.
33. 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, World Wide Web Consortium (W3C), Feb. 2006. <http://www.w3.org/2001/sw/BestPractices/SE/ODA>.
34. M. Uschold and M. Gruninger. Ontologies: Principles, Methods and Applications. *Knowledge Engineering Review*, 11(2):96–155, June 1996.
35. Y. Wand and R. Weber. On the deep structure of information systems. *Information Systems Journal*, 5(3):203–223, 1995.
36. J. Warmer and A. Kleppe. Building a Flexible Software Factory Using Partial Domain Specific Models. In J. Gray, J.-P. Tolvanen, and J. Sprinkle, editors, *Proceedings of the 6th OOPSLA Workshop on Domain-Specific Modeling (DSM'06), Portland, Oregon, USA*, volume 37 of *Computer Science and Information System Reports*, pages 15–22. University of Jyväskylä, Finland, Oct. 2006.