Developing Consistent and Modular Software Models with Ontologies

Robert Hoehndorf [a] Axel-Cyrille Ngonga Ngomo [b] Heinrich Herre [c]

[a] *Institute for Medical Informatics, Statistics and Epidemiology, University of Leipzig and Department of Computer Science, University of Leipzig and Department of Evolutionary Genetics, Max Planck Institute for Evolutionary Anthropology*

[b] *Department of Computer Science, University of Leipzig*

[c] *Institute for Medical Informatics, Statistics and Epidemiology, University of Leipzig*

Abstract. The development and verification of software models that are applicable across multiple domains remains a difficult problem. We propose a novel approach to model-driven software development based on ontologies and Semantic Web technology. Our approach uses three ontologies to define software models: a task ontology, a domain ontology and a top-level ontology. The task ontology serves as the conceptual model for the software, the domain ontology provides domain-specific knowledge and the top-level ontology integrates the task and domain ontologies. Our method allows the verification of these models both for consistency and ontological adequacy. This verification can be performed both at development and runtime. Domain ontologies are replaceable modules, which enables the comparison and application of the models built using our method across multiple domains. We demonstrate the viability of our approach through the design and implementation of a semantic wiki and a social tagging system, and compare it with model-driven software development to illustrate its benefits.

Keywords. Software engineering, formal ontology, ontology-driven design

1. Introduction

Current approaches to software development target at modular and reusable software. The development of such software requires both an understanding of the tasks that the software is supposed to perform and knowledge about the domain in which the software is applied. Software models provide a means for specifying these characteristics. They require a method both for making domain knowledge explicit and for integrating the domain knowledge with the conceptual model of the software. Current approaches to software are limited in this regard, as they are unable to separate conceptual and domain models. Therefore, software implemented using current approaches to software development can not be ported between domains without altering their conceptual model and consequently the whole software model.

We propose a method for developing software based on the interactions of three different kinds of ontologies: the conceptual model of the software called the *task ontology*, a *domain ontology* and a *top-level ontology*. The task ontology is an ontology for the problem domain, i.e., the problem that the software is intended to solve. The domain

ontology provides domain-specific knowledge for use by the software. The software can use the domain ontology to verify the ontological adequacy of the data it processes. The top-level ontology integrates these ontologies and allows for information flow between them. It also provides a means for integrating data from different domains.

We demonstrate the viability of this method with two case studies: creating a semantic wiki that guides users within a domain to enter correct knowledge and creating a collaborative tagging system that recognizes the types of tagged objects and adapts to them. We provide a comparison with model-driven software engineering and present the advantages of our approach.

2. Background

2.1. *Ontological foundations of conceptual modelling*

Conceptual modelling is concerned with the identification, analysis, design and description of both concepts and relations that are related to some domain of interest. This information is specified in a modelling language based on a set of meta-concepts forming the meta-model. Usually, conceptual modelling languages and the conceptual systems designed in these frameworks are evaluated based on their successful application, whereas the underlying meta-models are not further analyzed and evaluated. An ontological foundation of conceptual modelling goes a step further: it aims at a semantic reduction of the conceptual systems to a top-level ontology and its extensions in a principled way [1]. Ontologies, and in particular top-level ontologies, are rooted in methods of philosophy, logic and artificial intelligence, and they provide a framework for conceptual modelling [2].

Top-level ontologies can be used to evaluate the correctness of a conceptual model, but also to develop guidelines for designing conceptual models. In recent years, these problems were studied by several authors [2–4]. The approach presented here takes additional steps towards establishing the role of ontologies in the design of both conceptual models and software systems.

2.2. *Ontology*

In computer science, an ontology refers to an explicit specification of a conceptualization of a domain [5]. A conceptualization contains the basic categories and relations used in a language to describe a domain. An ontology specifies the intension of these basic categories and relations through a set of conditions, which are presented as axioms in some formalism. Based on their generality and scope, different types of ontologies can be distinguished: top-level, core and domain ontologies.

A top-level ontology contains categories relevant to every domain of reality [6]. Examples of these categories are *Process*, *Object* or *Category*. Several top-level ontologies are available for use, like the Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) [7] or the General Formal Ontology (GFO) [6]. Each possesses different features that determine their suitability for different applications. The examples presented here are based on the GFO.

GFO is a top-level ontology integrating objects and processes [6]. It is a layered ontology [8] that includes a theory of higher-order categories. These allow statements

about categories and the interrelations between them to be asserted, i.e., categories can be treated as the objects of a domain. The GFO also employs a multi-categorical approach, distinguishing between universals, concepts and symbols [9].

A core ontology [10] formally describes the basic categories within a domain. It makes the nature of a domain precise. At least two views on core ontologies must be distinguished. The first assumes that a core ontology contains the most general categories with respect to a taxonomy on a domain. These categories specialize the categories of a top-level ontology. The second view assumes that a core ontology is a formal theory of the central or principal categories within a domain and their derivations. These central categories and their derivations describe what the domain is about.

A domain ontology contains domain-specific types, relations and constraints. They can be founded in a top-level or a core ontology. An example of a domain ontology is the Pizza Ontology[1].

3. A method for ontology-based software engineering

In this section, we illustrate a method for developing software systems based on ontologies. This method uses three ontologies to define software models: a task ontology, a domain ontology and a top-level ontology. The task ontology serves as the conceptual model for the software, the domain ontology provides domain-specific knowledge and the top-level ontology integrates the task and domain ontologies.

We use the simple example of an ontology editor throughout this section to illustrate our method. The ontology editor can be used to create ontological categories and relations between these categories. The relations can have an arbitrary number of arguments. Every category can have instances.

3.1. Three ontology method

Our method is based on ontological foundations of conceptual modelling for software engineering [2] and Semantic Web technology. In order to capture and formalize the meaning [11] of the conceptual model, i.e., to make its ontological commitment explicit, we argue for the use of ontologies as part of software models. Then, software that is based on these models can access its own ontological commitment. Recent achievements in Semantic Web technology, in particular libraries for RDF[2] and OWL [12, 13], as well as expressive and feature-rich description logic reasoners [14, 15], make the realization of this goal possible.

The method we propose consists of three steps. First, an ontology is created as the conceptual model for the software. For this step, the results of research on ontological foundations of conceptual modelling can be employed, e.g., ontological foundations for UML [16]. Consequently, developing a shopping software requires the creation of an ontology for the shopping domain, developing a wiki requires a wiki ontology, developing a tagging software requires a tagging ontology, and so on. We call the resulting ontology

[1] http://www.co-ode.org/ontologies/pizza/
[2] http://librdf.org

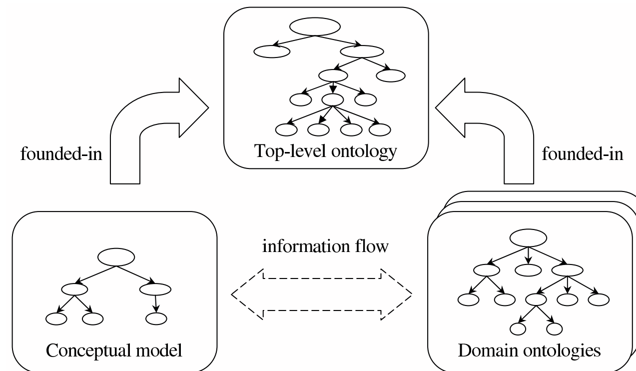

Figure 1. Three-ontology method: conceptual model, core or domain ontology and the top-level ontology integrating both. The domain ontology provides domain-specific knowledge and the top-level ontology provides a means for allowing information flow between the domain ontology and the conceptual model.

the *task ontology*[3][4]. Second, a top-level ontology is used as the foundation for the task ontology. The foundation can be established using a method of ontological reduction and mapping[5] [1, 18]. Third, a domain ontology, founded in the same top-level ontology as the task ontology, is used to provide domain-specific knowledge that is used within the software. Figure 1 illustrates the interactions between these ontologies. The software implementing this method uses the task and the top-level ontologies to specify an interface for using entities from domain ontologies. Therefore, domain ontologies are replaceable modules in this architecture.

All three ontologies must be available in a decidable logic such as OWL-DL [19] in order to be used by the software during runtime. Furthermore, it is necessary for the software to make use of a reasoner as a means for accessing and processing the ontologies. The application of our method leads to ontology-driven software [17], i.e., software that uses ontologies as a central part of their operation. Based on these considerations, we describe how the combination of Semantic Web technology together with the three ontologies contributes to the development of robust and reusable software and software models.

3.2. Role of task ontology

The task ontology is the conceptual model of the software; it contains the types to which the software can react, i.e., a conceptualization of the problem that the software is supposed to solve. These types are usually directly implemented and used in the software, e.g., as classes or modules. Directly using an ontology as the conceptual schema together with a reasoner brings with it a series of benefits. Notably, the reasoner can answer intensional queries about the conceptual schema and make the answers available to the soft-

[3]The task ontology is an ontology for the problem that the software solves, or the task it is supposed to perform. It specifies the conceptual model of the software. It is different from the "task ontology" in [17], which is an ontology of tasks.

[4]We use *task ontology* and *conceptual model* interchangeably throughout the remainder of the paper.

[5]Although considerable research has been invested in this area, no simple solution to this problem is known to us. The foundation is usually carried out manually by the ontology designer. In this paper, we assume a means for establishing this foundation as given.

ware. Furthermore, it can verify the consistency of the data with the conceptual schema during the runtime of the software. Due to the ontological foundation of the conceptual model and the possibility to query this model, the software system has access to *real-world* types and knowledge. This is a direct application of Semantic Web technology to the task of software development.

The specification of a task ontology does not suffice to completely specify a software model because it does not contain information pertaining to the application of such an ontology-driven software within a specific domain. Application to a domain necessitates the availability of additional knowledge about the types, relations and constraints that govern the domain. This is usually not captured in the conceptual model of the software. For example, the conceptual model of our ontology editor contains categories for *Relation* and *Category*, but not specific relations like *part-of* or specific categories such as *Baking* or *Pizza*. We will use the domain ontology to address this issue.

Furthermore, when applying software within different domains, a principled way for exchanging information between these applications is beneficial. In order to preserve the individual semantics of statements within each domain, the ontological status of the types within domains must be made precise and transferred together with the data. In the case of our ontology editor, consider one application to the domain of online shopping and another to the domain of pizzas. In order to integrate information contained in both domains of application, meta-information about the types that are used must be made explicit. In the particular example of the ontology editor, the categories that represent relations and categories must be made explicit[6]. This problem will be addressed by the top-level ontology.

Finally, it may occur that domain-specific types and relations are related in specific ways to types used in the conceptual model. For example, our ontology editor may be used together with a domain ontology that contains a category of relations. These must be used by the editor both as categories and as relations. The simplest relation between domain categories and categories from the conceptual model is "equivalence", when the domain of application contains types that are used in the conceptual model itself. To support the interaction between a software's conceptual model and the model of a domain, principles for the interaction between domain-specific knowledge and the conceptual model must be made explicit in way that allows for information flow between both. We will use the top-level ontology for this purpose.

3.3. Role of domain ontology

To support the goal of developing modular, domain-independent software models, we use domain ontologies to adapt the software to a domain. The domain ontology contains relations, types and laws specific to the domain.

The domain ontology is an exchangeable module both in the software model and the software architecture. To apply the software system in a domain, a domain ontology must be used to provide domain-specific knowledge. The types of the domain ontology can be used by the software for tasks such as asking intensional queries about the do-

---

[6]There are two readings of "category" in this sentence. The first refers to the categories that are used in the conceptual model of the software. The second refers to their instances. In GFO, these are instances of the *Category* category. In ontologies without higher-order categories, these are sub-categories of the *Entity* category.

main of application and verifying the ontological adequacy of data according to domain knowledge.

In the example of the ontology editor, arguments of relations may be filled only by instances of certain domain-specific categories. Which domain-specific types' instances can fill which arguments can be found out by an intensional query on the domain ontology. Furthermore, when an instance of a relation category is also declared as the instance of a category disjoint with relation, an exception can be declared and appropriate error handling performed. Our improves reusability over approaches such as MDE. For example, transforming the model of an online shop into an online library require only a change in the domain ontology and its mappings to the other elements of our method.

Two points remain open: how are the conceptual model of the software and the domain ontology related, and how can two software systems with the same conceptual model but using different domain ontologies exchange or integrate their data. These problems are addressed by the third component in our method, the top-level ontology.

3.4. Role of top-level ontology

For integrating the task and domain ontologies, we use a top-level ontology. It provides an ontological foundation for the categories of both the domain and task ontology. Since the task ontology is founded in a top-level ontology, the task ontology's entities can interrelate with other ontological entities, including those included in the domain ontology.

Through the top-level ontology, information can flow [20] in both directions between the task and the domain ontology. First, the types and relations of the domain ontology are combined with elements of the conceptual model through the top-level ontology. This extends the conceptual model with domain-specific concepts, relations and axioms. Our ontology editor can then access the categories and relations that are available within the domain ontology.

Second, domain-specific knowledge may entail the existence of instances of categories which belong to the conceptual model. This can introduce new information to which the software must react. For example, creating an instance of a domain ontology category that is a sub-category of *Relation* must entail the creation of a new relation within the ontology editor.

4. Semantic Wiki: the BOWiki

Our method has been applied to the development of the BOWiki [21], an ontology-based semantic wiki. In this section, we illustrate how our method lead to the development of a modular and extensible implementation.

4.1. Problem statement

The goal was to develop a wiki for the structured annotation of data. The annotation of data refers to the association of categories from ontologies with a piece of information. This type of annotation is widely used in parts of biology, where data is annotated to categories from biological ontologies. The wiki must allow for the creation of relations between entities. Relations can have arbitrary arity. In order to distinguish the arguments

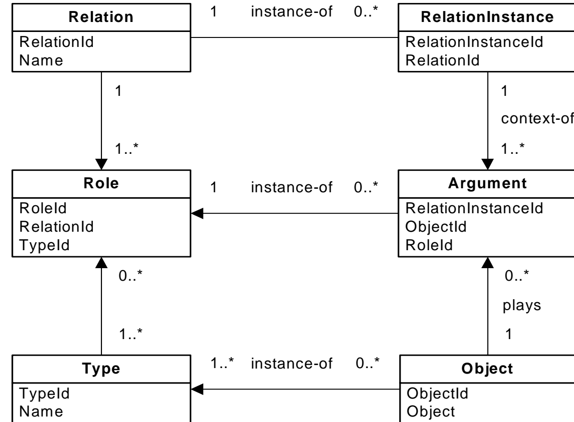

Figure 2. Conceptual model of the BOWiki in UML.

of these relations, they are constructed out of relational roles [22] (in their simplest form, named argument slots).

In order to provide a form of quality control for the knowledge captured in the wiki, not every object can play every role in every relation. Roles can only be played by objects of a certain *type*. The available types depend on the domain in which the wiki software is applied. Relations may also depend on the domain of discourse, but for integrating knowledge bases using the wiki software, new relations can be declared. The available types within the wiki are imported from an OWL-DL ontologies.

During the set-up of the wiki software, background knowledge about the domain can be provided, and the wiki uses this knowledge to verify some aspects of its content. In addition, we want to provide a means to assert interrelations between more than two entities by using n-ary relations. Our primary application is biology, where we aim at applying the wiki to the annotation of genes and gene products to terms from ontologies. However, the BOWiki software is intended to be domain-independent.

4.2. Conceptual model

The BOWiki is an extension of the MediaWiki software. It treats wikipages as instances and allows the assertion of relations between these instances. Relations have named argument slots (roles) that can be filled by objects of a specific type. Types are either datatypes or concepts from an OWL ontology (object types).

The basic categories that the BOWiki uses are relations, roles and types together with the instantiation relation, the *plays* relation that connects an object with a role, and the *role-of* relation that connects a role with a relation. Several restrictions can be formulated and used by the BOWiki to verify consistency, such as $Roles \sqsubseteq \exists roleOf.Relation$ or $Roles \sqsubseteq (= 1) plays.\top$. The BOWiki uses the Pellet description logic reasoner [15] to verify these constraints during runtime. The conceptual model of the BOWiki is the module of the GFO pertaining to relations and roles [22]. This module is described in Figure 2.

4.3. Top-level ontology

Because the conceptual model for the BOWiki is a part of the top-level ontology GFO, its ontological foundation in a top-level ontology is trivial. Due to its foundation in GFO, additional information is added to relations, roles and types. Relations and relational roles are concrete entities, i.e., they are in time and space. They are disjoint from other entities like processes, categories, or sets.

The BOWiki software uses both the GFO and its conceptual model in conjunction with a description logic reasoner. Therefore, additional constraints are provided by the use of a top-level ontology. If axioms from either ontology are violated, an edit in the BOWiki is rejected and the user notified.

4.4. Domain ontology: GFO-Bio

Domain-specific knowledge can influence the conceptual model of the BOWiki application by prodiving types of categories, relations and roles, that are interpreted accordingly by the BOWiki.

We want to apply the BOWiki for the annotation of data to categories of biological ontologies. For this purpose, we want to describe objects of specific biological types within the BOWiki. For example, the category of FOXP2 proteins is related to the category of *Language development* by the *participates in* relation. In this example, the types *Category*, *FOXP2 protein* and *Language development* appear and a binary relation *participates in*. The types *Protein* (a material structure) and *Language development* (a process) are disjoint. The relation *participates in* has two roles which we call *subject* and *object*, and each role may only be played of entities of a specific type (domain and range restrictions). We use the biological core ontology GFO-Bio [23] for this application.

5. Collaborative Tagging

Tagging refers to the association of a set of keywords with some object. Collaborative tagging enables multiple users to individually tag objects and share the tagged objects or the tags for these objects. There is an enormous number of available systems for the collaborative tagging of entities. Users can tag movies (YouTube[7]), pictures (Flickr[8]), Websites (del.icio.us[9]) or documents (CiteULike[10]). Depending on the *type* of tagged object, different tagging platforms are implemented. The type of tagged object determines the attributes that are stored with it. For documents, these may be the author, date of publication, journal, etc. For a webpage, it may be its URL, for photographs the type of camera used to take it or for movies the actors and director.

Depending on the type of tagged object, the tags may describe different aspects or facets of it. Some tagging systems support the use of facets: tags can be associated to different aspects of the tagged object. Often, several default facets like “theme” or “topic” are used. While these facets are common to many tagging systems, several possible facets

[7] http://www.youtube.com
[8] http://www.flickr.com
[9] http://del.icio.us
[10] http://www.citeulike.org

depend on the type of tagged object: videos may not only have a topic, but also temporal duration or temporal parts. Photographs have color-schemes. Molecules have functions, structural parts, shape and weight.

We describe a collaborative tagging system[11] that allows for tagging objects of different types. Depending on the type of tagged object, different information about the object is stored. In addition, tags can be associated to *facets* of objects. Some facets depend on the type of tagged object, while others are applicable to any tagging action.

We outline the tagging ontology of [24], which forms the foundation of the tagging software discussed here. It is based on [25] and the GFO [6].

A basic entity in the tagging domain is *Tag*, which is the role played by the string a tagger enters during a tagging action. The tag is a concrete individual. It instantiates a special kind of category, a *Symbol structure*. The tag is a *token of* the symbol structure.

The tag is associated with an object. This object can be any entity. While the object referred to by a URI is often identified with its URI [25, 26], in the analysis of tagging, it becomes important to distinguish between the object *described by* a URI and the URI itself. A tag can relate to either of these, and the nature of this relationship differs. Therefore, [24] distinguishs two kinds of entity, an information object containing information about some other entity, and the entity that is described by the information object.

It is assumed in [24] that tags are always associated to objects, not information resources describing them. In order to specify the way that some entity relates to whatever is denoted by a tag, *facets* are introduced. Facets are relationships that an entity can have to other entities. For example, physical objects can have a *part-of* facet, a *participates-in* facet, but also facets relating it to categories, like an *instance-of* facet. According to the tagging ontology, every entity can be denoted by some other entity (an information resource). When the denotation itself is used as a facet and combined with relations available for information resources, entities can be tagged so that the meaning of the tag relates to the information resource describing the entity.

Tagging is an intensional act, i.e., it involves a conceptualization of the tagger. In particular, not every instance of the same symbol structure is used to denote the same thing. Therefore, different taggers associate different concepts with tokens of a symbol structure. For example, the tag *bank* can refer to many different entities, depending on the background of the tagger. The domain and range restrictions of the relations used to construct facets for tagged entities can also be used to clarify the different meanings of tags depending on the tagger. For a full discussion of the tag ontology, see [24].

While this forms a comprehensive core ontology for the tagging domain, most tagging systems do not use all of them. For example, faceted tagging systems are rarely employed, concepts are almost never used and the distinction between an entity and the information resource that describes it is seldom explicated. However, this tagging ontology provides a means for analyzing collaborative tagging systems. Even if only a fragment of the ontology is used as the conceptual schema of a concrete implementation, the ontology can be used to share information with other tagging systems, if they employ a similar schema.

A domain ontology like GFO-Bio can be used to configure the tagging software. In particular, it can be used to generate the facets applicable to the tagged object, and the properties of the tagged object. The domain ontology provides the knowledge about the entities that play the role of the tagged object in the tagging relation.

---

[11] http://bioonto.de/pmwiki.php/Main/CollaborativeTaggingSystem

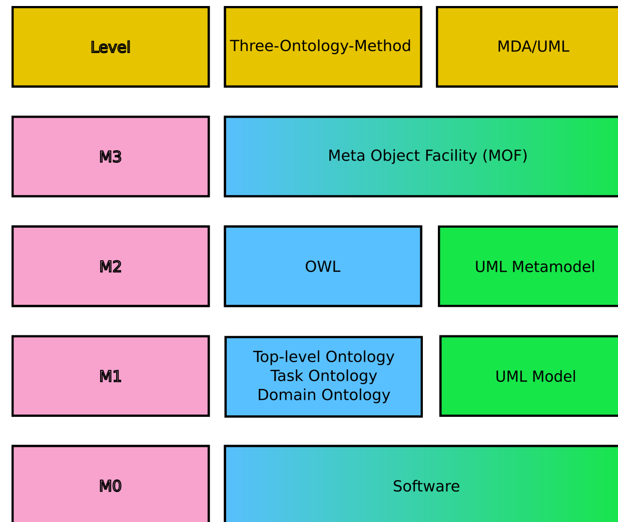

Figure 3. Comparison of our method with MDE. The four levels of abstraction in MDE are shown on the left. The three main components of our method are situated on M1, the model level. We use OWL in M2 as the meta-model. Model-driven architecture with UML is shown in the right. Both MDA and our method focus primarily on the levels M0 to M2. The level M3, the meta-metamodel, can be filled by the Meta Object Facility (MOF) [31] in both cases. This requires a MOF specification of OWL.

6. Comparison with Model-Driven Software Development

A recent development in the area of software development is centered around the use and development of models. Models are reusable artifacts and "provide a unique opportunity to mitigate complexity, improve consumability, and reduce time to market" [27]. Model-driven approaches such as Model-Driven Engineering (MDE) [28], Model-Driven Architecture (MDA) [29] and Model-Driven Software Development (MDSD) [30] aim at developing formal conceptual models for software and using them directly to generate code for software applications. In MDE, software models are specified using formal languages such as UML. A wide range of tools is available to support the generation of code from these models.

The method we propose is similar to MDE in some aspects. Both in MDE and our method, conceptual modelling plays a prominent role in the design and development of software systems. These can be used in both cases to automatically generate code for the designed applications. However, we use OWL as our meta-model while MDE is often carried out using UML. Therefore, using our method leads to semantically richer and modular models.

Our method goes beyond the current state of the art in MDE. We consider the following features the most outstanding advances over MDE: first, it provides an ontological foundation of the conceptual and domain models and consistency verification; second, it enforces a strict division between the conceptual and domain models; third, it allows for intensional queries and consistency checks during runtime.

The resulting model consists of the three components illustrated in Figure 3. When these models are formalized in a decidable fragment of OWL such as OWL-DL, automated reasoners can be employed to verify the model's consistency. This can be per-

formed for each component of the model individually, establishing the consistency of the top-level ontology, the conceptual model and the domain ontology. When combining the top-level ontology and the conceptual model or the top-level ontology and the domain ontology, inconsistencies can arise. These can be automatically detected and subsequently eliminated. The detection of inconsistencies assures the consistent foundation of both the conceptual model and the domain ontology in the top-level ontology. Finally, the consistency of all three model components together with their foundations in the top-level ontology is verified. As a result, the compatibility of the ontological commitments of the conceptual model and the domain ontology is ensured.

The division between the conceptual model of the software and the model for the domain to which the software is being applied leads to highly modular software models and thus to modular software systems. The conceptual model together with the top-level ontology provides an interface for integrating domain models in the form of domain ontologies. Such ontologies are available for many domains. For use with our method, these ontologies must be founded in the top-level ontology and be consistent with the conceptual model. This allows reusing domain ontologies for developing software models. Additionally, the part of the software model consisting of the conceptual model and the top-level ontology can be used across multiple domains.

The last major advancement over model-driven approaches lies in the availability of the software model during the runtime of the software. This can be used within the software system to verify the consistency of instances with both the conceptual model and the domain ontology during runtime. This can be used to verify constraints on data that is processed by the software. Furthermore, automated reasoners can be employed by the software during runtime to perform intensional queries on the software system's model itself. As such, the software has access to the types, relations and constraints that it can process. This could, for example, be used to implement semantic web services that automatically derive the types they accept as input or produce as output. These types can depend either on the conceptual model of the service, on the top-level ontology, on the domain ontology or a combination of these three components.

7. Discussion

The method we describe extends earlier work in formal ontologies in information systems [17] and applies it to software engineering. With the recent progress in Semantic Web technology, it is now possible to apply the method we describe to large-scale software development. To illustrate our method, we discussed the implementation of a semantic wiki and a collaborative tagging system.

A main advantage of our method is the use of a domain ontology as a module in a software model. This enables the construction of modular models that can be applied across multiple domains while preserving comparability and consistency. Using Semantic Web technology like description logic reasoners and OWL ontologies in the software systems yields a means for verifying the model's consistency and the ontological adequacy of stored and processed data. The foundation in a top-level ontology like the GFO [6] leads to ontologically well-founded models and makes the ontological commitment of these models explicit. Together with automated description logic reasoners, intensional queries both over the software's conceptual model as well as the domain ontology become possible.

There is a similarity between our method and Model-Driven Engineering [32] (MDE): in both methods, the conceptual model and the model of a domain play a central role in the software development process. In model-driven engineering, however, the conceptual model of a software system is used to generate code in various levels of detail. In our method, the model can be used during runtime for verifying the integrity of data and the internal state of the software. It is conceivable, however, that the software model generated according to our method is used in a model-driven engineering process to generate code. This code could then be generated in such a way that it automatically includes an OWL reasoner in the software code, and apply the same model that was used to generate the code for the verification of data during runtime.

When implementing software using the approach described here, the software must employ a reasoner for many of its internal operations. Although the performance of reasoners capable of processing expressive languages like OWL-DL has improved in recent years, reasoning remains a complex problem. In particular using a reasoner for applications that require high performance may currently be unfeasible.

8. Conclusion

We described a method for developing flexible, modular and consistent software models based on the interaction of three kinds of ontologies. The method reuses research in ontological foundations of conceptual modelling. It is based on the use of ontologies as conceptual models for software systems and the separation of the software's conceptual model from the model of a domain of application. We proposed to design software models in a way that enables using a domain ontology as a replaceable module. The interrelations between the domain ontology and the software's conceptual model are established using a top-level ontology. This permits applying parts of the model across multiple domains.

The method we described has been successfully applied to develop a semantic wiki and a semantic collaborative tagging system. We showed how our approach compares with model-driven engineering. In particular, we illustrated how our method exceeds model-driven engineering with respect to domain-dependence and the verification of the developed models both for consistency and ontological adequacy. Finally, we discuss how to integrate our method with model-driven engineering.

Acknowledgements

We thank Janet Kelso for valuable comments on this paper.

References

[1] Herre, H., Heller, B.: Semantic foundations of medical information systems based on top-level ontologies. Knowledge-Based Systems **19**(2) (2006) 107–115

[2] Guizzardi, G.: Ontological Foundations for Structural Conceptual Models. Volume 015 of Telematica Instituut Fundamental Research Series. Telematica Instituut (2005)

[3] Evermann, J., Wand, Y.: Towards ontologically based semantics for UML constructs. In: ER '01: Proceedings of the 20th International Conference on Conceptual Modeling, London, UK, Springer-Verlag (2001) 354–367
[4] Wand, Y., Storey, V.C., Weber, R.: An ontological analysis of the relationship construct in conceptual modeling. ACM Trans. Database Syst. **24**(4) (1999) 494–528
[5] Gruber, T.R.: Toward principles for the design of ontologies used for knowledge sharing. International Journal of Human-Computer Studies **43**(5-6) (1995) 907–928
[6] Herre, H., Heller, B., Burek, P., Hoehndorf, R., Loebe, F., Michalek, H.: General Formal Ontology (GFO) – A foundational ontology integrating objects and processes [Version 1.0]. Onto-Med Report 8, Research Group Ontologies in Medicine, Institute of Medical Informatics, Statistics and Epidemiology, University of Leipzig, Leipzig (2006)
[7] Masolo, C., Borgo, S., Gangemi, A., Guarino, N., Oltramari, A.: WonderWeb Deliverable D18: Ontology library (final). Technical report, Laboratory for Applied Ontology – ISTC-CNR, Trento (Italy) (2003)
[8] Herre, H., Loebe, F.: A meta-ontological architecture for foundational ontologies. In: On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE. Springer Verlag (2005) 1398–1415
[9] Gracia, J.J.E.: Metaphysics and Its Task: The Search for the Categorcal Foundation of Knowledge. SUNY Series in Philosophy. SUNY Press (1999)
[10] Valente, A., Breuker, J.: Towards principled core ontologies. In Gaines, B.R., Musen, M.A., eds.: Proceedings of the 10th Knowledge Acquisition Workshop (KAW'96), Banff, Alberta, Canada, Nov 9-14. (1996) 301–320
[11] Guarino, N.: The ontological level. In Casati, R., Smith, B., White, G., eds.: Philosophy and the Cognitive Sciences. Hölder-Pichler-Tempsky, Vienna (1994)
[12] Carroll, J.J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K.: Jena: Implementing the Semantic Web recommendations. Technical Report HPL-2003-146, Hewlett Packard, Bristol, UK (2003)
[13] Horridge, M., Bechhofer, S., Noppens, O.: Igniting the owl 1.1 touch paper: The owl api. In: Proceedings of OWLEd 2007: Third International Workshop on OWL Experiences and Directions. (2007)
[14] Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: System description. In Furbach, U., Shankar, N., eds.: Automated Reasoning: Proceedings of the Third International Joint Conference, IJCAR 2006, Seattle, Washington, USA, Aug 17-20. Volume 4130 of Lecture Notes in Computer Science., Berlin, Springer (2006) 292–297
[15] Sirin, E., Parsia, B.: Pellet: An OWL DL reasoner. In Haarslev, V., Möller, R., eds.: Proceedings of the 2004 International Workshop on Description Logics, DL2004, Whistler, British Columbia, Canada, Jun 6-8. Volume 104 of CEUR Workshop Proceedings., Aachen, Germany, CEUR-WS.org (2004) 212–213
[16] Guizzardi, G., Herre, H., Wagner, G.: Towards ontological foundations for UML conceptual models. In: On the Move to Meaningful Internet Systems, 2002 - DOA/CoopIS/ODBASE 2002 Confederated International Conferences DOA, CoopIS and ODBASE 2002, London, UK, Springer-Verlag (2002) 1100–1117
[17] Guarino, N.: Formal ontology and information systems. In Guarino, N., ed.: Proceedings of the 1st International Conference on Formal Ontologies in Information Systems, FOIS'98, IOS Press (1998) 3–15
[18] Heller, B., Herre, H., Lippoldt, K.: Domain-specific concepts and ontological reduction within a data dictionary framework. In: Data Integration in the Life Sciences. Springer Verlag (2004) 47–62
[19] McGuinness, D.L., van Harmelen, F.: OWL Web Ontology Language overview. W3C recommendation, World Wide Web Consortium (W3C) (2004)
[20] Schorlemmer, M., Kalfoglou, Y.: On semantic interoperability and the flow of information (2003)
[21] Hoehndorf, R., Prüfer, K., Backhaus, M., Herre, H., Kelso, J., Loebe, F., Visagie, J.: A proposal for a gene functions wiki. In Meersman, R., Tari, Z., Herrero, P., eds.: Proceedings of OTM 2006 Workshops, Montpellier, France, Oct 29 - Nov 3, Part I, Workshop Knowledge Systems in Bioinformatics, KSinBIT 2006. Volume 4277 of Lecture Notes in Computer Science., Berlin, Springer (2006) 669–678
[22] Loebe, F.: Abstract vs. social roles – Towards a general theoretical account of roles. Applied Ontology **2**(2) (2007) 127–158
[23] Hoehndorf, R., Loebe, F., Poli, R., Herre, H., Kelso, J.: GFO-Bio: A biological core ontology. Applied Ontology (2008) forthcoming.
[24] Uciteli, A.: Ontologien und kollaborative Taggingsysteme. Master's thesis, Department of Computer

Science, University of Leipzig (2008) forthcoming.
[25] Newman, R.: Tag ontology design. http://www.holygoat.co.uk/projects/tags/ (2005) Last accessed: Nov 20, 2007.
[26] Pepper, S., Schwab, S.: Curing the web's identity crisis: Subject indicators for rdf. In: Proceedings of the XML conference 2003. (2003)
[27] Larsen, G.: Model-driven development: Assets and reuse. IBM Systems Journal **45**(03) (2006) 541 – 554
[28] Bézivin, J.: On the unification power of models. Software and Systems Modeling **4**(2) (2005) 171–188
[29] Mellor, S., Scott, K., Uhl, A., Weise, D.: MDA Distilled: Principles of Model-Driven Architecture. Addison-Wesley, Boston (2004)
[30] Stahl, T., Voelter, M.: Model-Driven Software Development. John Wiley & Sons, Ltd (2006)
[31] Object Management Group: Omg's metaobject facility. http://www.omg.org/mof/ (2008)
[32] Schmidt, D.C.: Guest editor's introduction: Model-driven engineering. Computer **39**(2) (2006) 25–31