Bridging Software Languages and Ontology Technologies

Tutorial Summary

Fernando Silva Parreiras
Tobias Walter
Institute for Web Science and Technologies, University of Koblenz-Landau
D-56070 Koblenz, Germany
{parreiras,walter}@uni-koblenz.de

Christian Wende
Institute for Software- and Multimedia-Technology, Dresden University of Technology
D-01062 Dresden, Germany
c.wende@tu-dresden.de

Edward Thomas
Department of Computing Science, The University of Aberdeen
Aberdeen AB24 3UE
e.thomas@abdn.ac.uk

Abstract

Current model-driven development approaches allow for a more productive way of developing software systems. However, building tools and languages for software development still suffer a neglect of semantics in modeling and metamodeling.

An interest to strengthen semantics in modeling and metamodeling that gained scientific and commercial attention is the integration of ontology technology and software development. Ontology formalisms for consistency validation and dynamic classification as well as semantic web technologies for enabling shared terminologies and automated reasoning provide means for leveraging metamodeling and language engineering.

This tutorial summary (1) enlightens the potential of ontology and semantic web technology for modeling and metamodeling in software development, positioning it among modeling standards like UML, and MOF; and (2) illustrates ontology-enabled software development with real application scenarios in areas like software design patterns, domain-specific languages and variability management.

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

General Terms Design, Languages

Keywords Semantic Web, Ontology Technology, Software Languages, UML, DSL, Model-Driven Development

SPLASH'10, October 17–21, 2010, Reno/Tahoe, Nevada, USA.

1. Introduction

Semantic web technologies comprises a stack of standard and tools using metadata, logic and ontology languages, i.e., languages to describe formally a domain of discourse. Among ontology languages, the Web Ontology Language (OWL) [17] is the most prominent for Semantic Web applications, providing a class definition language for ontologies.

Indeed, OWL provides important features complementary to class-based model design that improve software languages: it allows different ways of describing classes; it handles these descriptions as first-class entities; it provides additional constructs like transitive closure for properties; and it enables dynamic classification of objects based upon class descriptions.

OWL has been applied into software engineering for many years in the form of Description Logic languages to achieve improvements on the maintainability and extensibility [9]. For example, the knowledge encoded in OWL evolves independently of the execution logic, i.e., developers maintain class descriptions in the ontology and not in the software. Moreover, developers may use class descriptions to semantically query the domain. Semantic query plays an important role where shared terminologies, interoperability and consistency detection are required.

This tutorial summary addresses the following question: What are the features and services provided by ontology technologies that can be used to improve software languages? What are the applications of these features and services in software language engineering?

We organize this tutorial summary as follows: Section 2 enlightens the potential of ontology and semantic web technology for modeling and metamodeling of software languages, positioning it among modeling standards like UML, and MOF. Section 2 illustrates ontology-enabled software languages with real application scenarios in areas like software design patterns, domain-specific languages, business process modeling and variability management.

2. Ontologies in Software Languages

Class-based modeling languages (e.g. UML class diagrams or MOF) and OWL comprise some constituents that are similar in many respects like classes, associations, properties, packages, generalization and instances [11].

Nevertheless, OWL offers a more expressive and extensible manner of modeling data and provides flexible ways to describe classes and, based on such descriptions, it enables type inference. Indeed, OWL provides various means for describing classes, which may also be nested into each other such that explicit typing is not compulsory. One may denote a class by a class identifier, an exhaustive enumeration of individuals, property restrictions, an intersection of class descriptions, a union of class descriptions, or the complement of a class description.

For example, software developers can use reasoning services (Sect. 2.2) to *dynamically classify* objects based on conditions specified in class descriptions (Sect. 3.1). Language users can count on *explanation services* (Sect. 2.2) for debugging and learning domain concepts (Sect. 3.2). Lastly, software developers might want to use class descriptions to integrate software languages and use *query answering* for retrieving information of multiple languages (Sect. 3.6).

Non-class-based modeling languages requires ad-hoc transformations into OWL for taking advantage of reasoning services. For example, while OWL contains elements like classes, properties and individuals, a business process modeling language contains elements like process, task, gateways, activities, states, etc.

Applications of OWL ontologies in those cases usually relies on *satisfiability checking* reasoning service to verify whether the integrity of model elements is fulfilled in all possible states. For example, software product line developers can carry out *satisfiability checking* to verify whether every feature of the feature model is instantiable (Sect. 3.4). Process modelers can verify whether a specific process model refining an abstract one is valid, i.e., they can verify whether the execution set of the abstract process model holds for every execution set of the refined process model (Sect. 3.3).

2.1 Demystifying OWL

In this section, we compare how the Semantic Web and Software Engineering worlds differ in terms of their world assumption (respectively, open world versus closed world). We show how an ontology can, either in whole (as locally closed world) or in part (locally closed domain) be made to behave under the closed world assumption. We also look at negation as failure in the context of ontologies.

The Semantic Web adopts an open world assumption, this extends to the ontology language OWL, and to the reasoners and tools which work with it. In simple terms, this means that any fact that cannot be proven to be true by the known data, cannot be assumed to be false unless it is directly contradicted by other data in the ontology. For example, given an OWL axiom which states that all cows eat only plants, and that sheep are animals (in DL syntax: $Cows \sqsubseteq \forall eats.Plant$ and $Sheep \sqsubseteq Animals$), asserting that some cow eats some sheep does not cause a contradiction, since it has not been stated anywhere that animals cannot also be plants ($Animals \sqsubseteq / Plants$). In fact, a reasoner will infer exactly this fact from the given knowledge.

It is possible in OWL to close the domain of a particular class, by asserting that that class is equivalent to exactly the set of all its members. This prevents the reasoner from inferring that any other individual (either a named individual, or an inferred individual) is a member of this class. A reasoner can be used to close the domain of a class or a property, affecting the outcome of various reasoning tasks on the ontology [16].

By closing the domain of an entire ontology, we can simulate a closed world assumption. This involves closing the domain of every class and property in an ontology. This compares to the closed world assumption in Software Engineering, but it has performance implications when closing the domain of a complex ontology.

An alternative to closed world assumption is to use negation as failure (NAF) in the Semantic Web context. The results from using NAF can differ from using a true (locally) closed world (or domain) approach, and a closed domain and NAF can be combined in ontology reasoning to allow default behaviour and other techniques to be performed (See [16]).

OWL2 comprises a family of description logic languages, called profiles, which offer different levels of expressiveness and tractability. By using different language profiles where different levels of performance and expressivity are required, and by exploiting quality-guaranteed transformations between these languages, hybrid reasoners such as TrOWL [16] can reduce the complexity of a given domain model, increasing performance.

1. *OWL2-QL* The primary application for *OWL2-QL* are for ontologies with large abox datasets. It supports storing and querying this data using an SQL database, using query expansion to perform complete query answering with respect to the semantics of OWL and the tbox of the ontology.
2. *OWL2-EL* OWL2-EL is a profile of OWL2 specifically designed for high performance TBox reasoning. It supports consistency, classification, class expression subsumption and instance checking in polynomial time (cf. 2NexpTime for OWL2-DL).
3. *OWL2-RL* OWL2-RL is the subset of OWL2 which amenable to expression as a set of rules. These rules are run over a set of ground facts (the concrete axioms given in the ontology file) to infer every axiom which can be derived from those facts under OWL2 RDF semantics.
4. *OWL2-DL* OWL2-DL can be regarded as the full expressivity of OWL2 available under OWL2 direct semantics.

It is the most expressive profile in OWL2 and it corresponds to the description logic $\mathcal{SROIQ}$.

OWL presents complementary constructs to UML and OCL, allowing, for example, property transitivity, which is very useful for querying large and complex models.

By targeting different profiles of OWL2 to different aspects of a reasoning problem, improved performance can be achieved. For example, taking a very large ontology, classifying it, and performing query answering in OWL2-DL may be too costly in time and memory. By transforming that ontology into an OWL2-EL and OWL2-QL representations, these tasks become much more tractable (PTIME-complete and NLogSpace-complete, instead of 2NEXPTIME-complete).

2.2 Reasoning Services for Model Design

OWL ontologies can be operated on by reasoners providing services like consistency checking, concept satisfiability, instance classification and concept classification.

Reasoners provide the following standard reasoning services:

Consistency Checking The reasoning service consistency checking checks if a given model is consistent with regard to their language metamodel.

Satisfiability Checking The satisfiability checking service finds all unsatisfiable concepts in a given language metamodel. A concept metamodel is unsatisfiable if it represents an empty set of instances.

Classification The classification service returns for a given instance a set of metamodel concepts which contain/describe the instance. The instance conforms to all concepts in the result of the classification service. The classification service is used to dynamically classify elements in models to find its most specific type.

Subsumption The subsumption checking service checks whether a concept is superconcept or subconcept of another given (anonymous) concept.

In addition, some reasoners provide a set of non-standard reasoning services:

Query Answering SPARQL [14] is a prominent query language for querying OWL ontologies. SPARQL queries are able to operate on both, the schema level (metamodel layer) and on the instance level (model layer).

Explanation Services The generation of explanations for inferences computed by a reasoner is now recognized as highly desirable functionality. If the inference leads to some inconsistency or unsatisfiable classes, the explanation service results some debugging relevant facts and the information how to repair the model.

3. Applications of Ontology Technologies in Software Languages

3.1 Improving General Purpose Software Design Patterns

In general, the Software Design Pattern deal with variation and delegation of concepts in software models. However, as already documented by [5], the Strategy Pattern has a drawback. The clients must be aware of variations and of the criteria to select between them at runtime. Hence, the question arises of how the selection of specific classes could be determined using only their descriptions rather than by weaving the descriptions into client classes.

In [15], we present an approach to decouple class selection from the definition of client classes by exploiting OWL-DL modeling and reasoning. It enables the identification of patterns integrating UML class diagrams with software design patterns and OWL like, for example, the Selector Pattern.

The application of the Selector Pattern presents the following consequences: (1) reuse – The knowledge represented in OWL-DL can be reused independently of platform or programming language; (2) flexibility – The knowledge encoded in OWL-DL can be modeled and evolved independently of the execution logic; and (3) testability – The OWL-DL part of the model can be automatically tested by logical unit tests, independently of the UML development.

The application of our approach can be extended to other design patterns concerning variant management and control of execution and method selection. Design patterns that factor out commonality of related objects, like Prototype, Factory Method and Template Method, are good candidates.

3.2 Ontology-Based Domain-Specific Languages

Domain-specific languages (DSL) are used to model and develop systems of different application domains. However, there is an agreement about the challenges faced by current DSL approaches: (1) tooling (debuggers, testing engines), (2) interoperability, (3) semantics, (4) learning curve and (5) domain analysis. Improving tooling enhances user experience while formal semantics is the basis for interoperability and formal domain analysis.

In [18, 19], we propose an Ontology Based DSL Framework that allows for defining DSLs enriched by formal class descriptions. It allows DSL designers to check consistency of the model and helps DSL users to verify and debug DSL programs by using reasoning explanation. Moreover, novice DSL users may rely on reasoning services to suggest domain concepts according to the language definition.

The integrated metamodel MOF+OWL allows for a formal and logical representation of the solution domain. Thus, DSL designers count on an expressive language that allows for modeling logical constraints over DSL metamodels. Consistency of metamodels and constraints can be checked by reasoners and inferences are clarified by explanation ser-

vices. The nature of the logical restrictions allowed by OWL enables progressive evaluation of DSL program consistency.

Moreover, the integration MOF+SPARQL enables DSL users to define SPARQL-like queries with the DSL metamodel to query objects in DSL programs. These queries are the interface between DSL user and reasoning services. For example, a DSL user may use a query defined in the DSL metamodel to query all classes that describe an object in the DSL program.

3.3 Ontology-Based Analysis in Variability Modelling

In software product lines [13], *feature models* are used to capture common and variable features in a family of related software products. Feature modeling originates from the FODA study [8]. In addition to simple relationships in hierarchical feature models as *mandatory*, *optional*, *alternative* or *exclusive-alternative* features several extensions were proposed. Most notably are propositional cross-tree relations (*requires*, *conflicts*) [1], cardinality-based features and groups [4], and feature attributes [3]. Using these constructs, feature models impose several constraints and relationships among features and, thus, specify all valid variants of the product line. During variant configuration *variant models* select a subset of the features in a product line's feature model to specify a concrete product.

Analysis in variability modelling checks the consistency of feature models, propositional feature terms and variant models w.r.t. the constraints and relationships specified for a concrete product-line and is an important area of research. For a recent, extensive overview of existing approaches on feature model analysis we refer to [2].

In this use case we discuss the application of ontology technology to validate feature and variant models and provide guidance in variant configurations. Based on existing approaches for the description logic based representation and analysis of feature models [20] we cover the following topics:

- Automated translation of hierarchical feature models with propositional cross-tree relations to OWL ontologies that represent their structural and semantics constraints,
- Automated translation of propositional feature terms and variant models to OWL ontologies,
- Application of OWL reasoning services to validate feature models, propositional feature terms, or variant models and to provide guidance in variant configuration,
- Integration of ontology-based analysis in variability modelling with the model-driven technology using the tool FeatureMapper [7],
- Exemplary applications of ontology-based variability modelling in SPLE, and
- Open challenges and issues in analysis of variability modelling.

3.4 Modeling and Querying Process Models with OWL

Process models capture the dynamic behavior of an application or system. In software modeling they are represented by graphical models like BPMN Diagrams or UML Activity Diagrams. The corresponding metamodels prove flexible means to describe process models for various applications. However, process models are often ambiguous with inappropriate modeling constraints and even missing semantics.

The process model in OWL gives an explicit description of the execution order dependencies of activities [6]. Hence, this information is used for process retrieval. A query describes the relevant ordering conditions like which activity has to follow (directly or indirectly) another activity. For instance a process that executes the activity $FillOrder$ before $MakePayment$ with an arbitrary number of activities between them, is given by the query process description $\exists TOT.(FillOrder \sqcap \exists TOT.MakePayment)$. The transitive object property TOT is used to indicate the indirect connection of the activities. The result are all processes that are subsumed by this general process description.

Besides ordering constraints, this semantic query processing allows the retrieval of processes that contain specialized or refined activities. For instance the result of the demonstrated query also contains all processes with subactivities of $FillOrder$ and $MakePayment$. The corresponding class expressions in the OWL model are specializations of the class expression given by the query expression. Finally, the usage of in the queries allows handling of modality for activity occurrences in a process, like a query that expresses that the activity $ShipOrder$ has to occur or might occur.

3.5 Metamodeling

The UML allows for capturing information about multiple views of systems like static structure and dynamic behavior. Since it is hard to capture all aspects of software into only one model, UML includes numerous types of diagrams to be used according to the software development task.

Since the semantics of UML constructs is textually described in the UML specification, it is hard to guarantee the same behavior across multiple implementers. Moreover, since UML enables multiple views of systems, it is important to have a consistent view over all UML diagrams. We have used the integration MOF+OWL [12] to model OWL descriptions at the metamodeling level. Metamodeling with OWL helps to disambiguate UML constructs and allow to specify logical constraints only textually described yet.

We have analyzed the different types of relationships in the UML2 Specification [10] and identified constraints that could benefit from our approach. For example, where property transitivity is required, e.g., in specifying constructs like Activity, State, StateMachine and Transition, our approach

allows for defining additional operations that are not easily expressible in OCL.

3.6 Enabling Linked Data Capabilities to Software Languages

In the software development process, there are standards for general-purpose modeling languages and domain-specific languages, capable of capturing information about different views of systems like static structure and dynamic behavior. In a networked and federated development environment, modeling artifacts need to be linked, adapted and analyzed to meet the information requirements of multiple stakeholders.

We propose an approach for linking, transforming and querying models expressed in MOF compliant languages, including OMG standards and domain-specific languages. We define structural mappings between MOF and OWL and propose the usage of semantic web technologies for linking and querying software models.

We show that the usage of OWL for specifying metamodels is a viable solution to achieve interoperability and shared conceptualizations. The role of OWL is not to replace MOF or the Object Constraint Language because OWL addresses distinct requirements, specially concerning networked environments. OWL should compose the spectrum of software modeling languages in a unified architecture.

4. Conclusion

In tutorial summary paper we present an overview of the main features of ontology technologies for software languages and exemplify this features and services with case studies being developed under EU STReP MOST that use ontology technologies.

References

[1] D. S. Batory. Feature models, grammars, and propositional formulas. In J. H. Obbink and K. Pohl, editors, *SPLC*, volume 3714 of *LNCS*, pages 7–20. Springer, 2005.

[2] D. Benavides, S. Segura, and A. Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Inf. Syst.*, 35(6):615–636, 2010.

[3] K. Czarnecki and C. Kim. Cardinality-based Feature Modeling and Constraints: A Progress Report. In *OOPSLA05 Workshop on Software Factories, October 17, 2005, San Diego, California, USA.*, 2005.

[4] K. Czarnecki, S. Helsen, and U. Eisenecker. Formalizing Cardinality-based Feature Models and their Specialization. *Software Process: Improvement and Practice*, 10(1):7–29, 2005.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley, Boston, MA, USA, 1995.

[6] G. Gröner and S. Staab. Modeling and Query Patterns for Process Retrieval in OWL. In A. Bernstein, D. R. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta, and K. Thirunarayan, editors, *International Semantic Web Conference*, volume 5823 of *LNCS*, pages 243–259. Springer, 2009.

[7] F. Heidenreich, J. Kopcsek, and C. Wende. Featuremapper: mapping features to models. In *ICSE Companion '08: Companion of the 30th international conference on Software engineering*, pages 943–944, New York, NY, USA, 2008. ACM.

[8] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Carnegie-Mellon University Software Engineering Institute, November 1990.

[9] D. L. McGuinness. Configuration. In F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors, *The Description Logic Handbook*, chapter 12, pages 397–414. Cambridge University Press, 2003.

[10] OMG. *Unified Modeling Language: Superstructure, version 2.1.2*. Object Modeling Group, November 2007.

[11] OMG. *Ontology Definition Metamodel (ODM) Version 1.0*. Object Modeling Group, May 2009.

[12] F. S. Parreiras and S. Staab. Using ontologies with UML class-based modeling: The Twouse approach. *Data & Knowledge Engineering*, In Press, Accepted Manuscript, 2010.

[13] K. Pohl, G. Böckle, and F. Van Der Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer-Verlag, 2005.

[14] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. Technical report, W3C, 2008.

[15] F. Silva Parreiras, S. Staab, and A. Winter. Improving design patterns by description logics: A use case with abstract factory and strategy. In *Modellierung 2008*, volume P-127 of *LNI*, pages 89–104. GI, 2008.

[16] E. Thomas, J. Z. Pan, and Y. Ren. Trowl: Tractable owl 2 reasoning infrastructure. In L. Aroyo, G. Antoniou, E. Hyvönen, A. ten Teije, H. Stuckenschmidt, L. Cabral, and T. Tudorache, editors, *The Semantic Web: Research and Applications, 7th Extended Semantic Web Conference, ESWC 2010, Heraklion, Crete, Greece, May 30 - June 3, 2010, Proceedings, Part II*, volume 6089 of *LNCS*, pages 431–435. Springer, 2010.

[17] W3C OWL Working Group. OWL 2 Web Ontology Language Document Overview. W3C Recommendation 27 October 2009, 2009.

[18] T. Walter, F. Silva Parreiras, and S. Staab. OntoDSL: An Ontology-Based Framework for Domain-Specific Languages. In *Model Driven Engineering Languages and Systems, 12th International Conference, MODELS 2009*, volume 5795, pages 408–422. Springer, 2009.

[19] T. Walter, F. Silva Parreiras, J. Ebert, and S. Staab. Joint Language and Domain Engineering. In *Proc. of 6th European Conference on Modelling Foundations and Applications , ECMFA 2010, Paris, France, June 15-18, 2010*, LCNS. Springer, 2010.

[20] H. H. Wang, Y. F. Li, J. Sun, H. Zhang, and J. Pan. Verifying feature models using OWL. *Web Semant.*, 5(2):117–129, 2007.