A Model-based Product-Line for Scalable Ontology Languages*

Christian Wende and Florian Heidenreich

Technische Universität Dresden
Institut für Software- und Multimediatechnik
D-01062, Dresden, Germany
{c.wende|florian.heidenreich}@tu-dresden.de

Abstract. Research in the area of semantic web brought up a plethora of languages to represent ontologies. They all differ in expressiveness and reasoning efficiency. Thus, the choice of a specific language means a trade-off between reasoning capabilities and performance. This paper outlines how techniques from product-line engineering can be combined with model-based language engineering to allow for organising ontology languages in a language family and configuring them for concrete use cases.

1 Introduction

Ontologies provide means for encoding knowledge about specific domains and often include reasoning rules that allow for deriving implicit knowledge. The manifold of domains that ontology languages are applied to led to a plethora of languages to represent ontologies. In this paper we focus on a subset of ontology languages based on the OWL2 standard [25] that provides an implementation for Description Logics (DLs) [1]. They share the common approach of representing knowledge using hierarchies of unary atomic *concepts* that are augmented with binary logical operators or *roles* to describe concept relationships. Expressiveness and reasoning efficiency is directly determined by the concrete binary operators a language provides [6]. Thus, the choice of a specific language means a trade-off between performance and reasoning capabilities. To achieve scalability of ontology languages both performance and functional requirements of a specific use case need to be consider.

Customising ontology languages from a language family has been identified to be a promising approach to address specifics of the use case they are applied to [27]. In addition, it has a number of other benefits: (1) Common language features can be reused among the language family members. (2) The family members are organised in a systematic way. (3) Specific expressiveness and reasoning requirements can be addressed by recombining existing language features. (4)

* This research has been co-funded by the German Ministry of Education and Research (BMBF) within the project feasiPLe and by the European Commission within the FP7 project MOST contract number 216691.

Ontology language evolution can be realised by contributing new features to the language family. (5) Language tooling (e.g., dedicated parsers, printers, editors, and reasoners) can be generated. To support a systematic approach for language customisation, we argue for combining techniques from product-line engineering and model-based language engineering. This paper contributes a systematic classification of ontology languages using the the paradigm of feature modelling. This enables feature-based customisation of OWL2 w.r.t. a specific use case. Second, we introduce a model-driven process to generate the tool infrastructure that is crucial for the adoption of the language variant in knowledge modelling.

The rest of this paper is structured as follows: Section 2 discusses the application of feature modelling to specify commonalities and variability among a family of OWL2-based ontology languages and explains how to use this variability information to select a language variant matching the needs of a specific use case. Section 3 discusses the generation of the tool infrastructure that is crucial for the application of a custom language variant. We introduce a model-driven process to automatically generate a parser, printer, and editor for a given language variant. For the sake of complexity we consider the generation of semantic tooling (e.g. reasoners) out of the scope of this paper. However, even with existing reasoners reasoning efficiency can be scaled by purely syntactic language adaptation [8, 13, 23]. We conclude and present future work in Section 5.

2 Feature-Based Ontology Language Configuration

The syntactic and semantic expressiveness of ontology languages is described inductively by the DLs [1] constructors they provide for knowledge representation [6]. The connection between reasoning characteristics of an ontology language and its logical constructors motivates a guidance of language configuration by means of singular logical constructors. Since constructors are interdependent and interact, it is necessary to specify dependencies and relations between them. This can be done using the paradigm of feature modelling.

The feature model depicted in Fig. 1 is based on the constructors used in DLs to describe the expressiveness of ontology languages. It describes commonalities and variability in our OWL2-based family of ontology languages. Every feature represents a single constructor by its textual name and the letter used in the usual naming conventions for DLs (for an overview see [1]). In addition, a cardinality is given for every language feature. A feature connected by a line ending with a filled circle represents a mandatory feature and is used in every OWL2 language variant. Empty circles identify optional features.

All possible feature combinations span the variation space for our language family. A concrete selection of features from this variation space describes a specific ontology language variant. Its simplest member can be built using only the mandatory features (Concepts, Top, Bottom, Intersection, AtomicNegation and ValueRestriction). It corresponds to the minimal Attributive Language $\mathcal{AL}$ that is considered the base of our desciption logics language family. By including for example the optional features $\mathcal{R}+$, $\mathcal{C}$, $\mathcal{H}$, $\mathcal{O}$, $\mathcal{I}$, $\mathcal{N}$, and $(\mathcal{D}+)$ one could

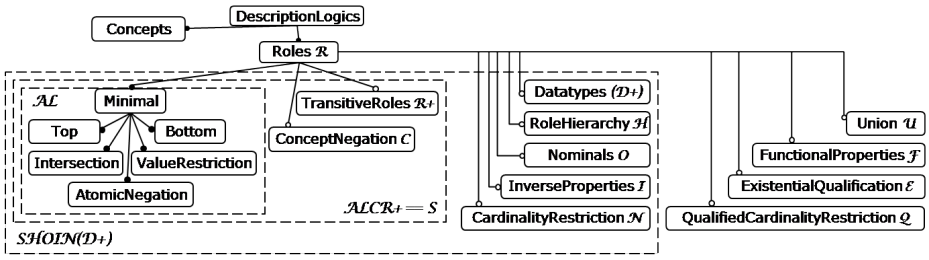

Fig. 1. Feature model that describes the variability space of ontology languages

configure the language variant $\mathcal{SHOIN}(\mathcal{D}+)$ with the expressiveness of OWL DL [21]. When features or feature combinations are annotated with their implications on reasoning efficiency (as studied in [6]), this feature model can be used to guide the customisation process regarding language expressiveness and efficiency. In addition, one can identify gaps in the language hierarchy and fill them by configuring constructors or by adding new logical constructors.

This feature-based modularisation of languages introduces a foundation for making reasoning technology more scalable: Optimised language variants can be configured that take the concrete expressiveness and efficiency requirements of a specific use case into account.

3 Model-based Ontology Language Engineering

There are three driving forces that made us address the problem of building the language family in a model-based way. First, modelling techniques offer support for transformation of models into other representations, e.g., reduced models or more specific models. This is particularly needed when deriving a concrete language from the language product-line. Second, models can be easily used for code generation, which is needed to automatically generate tool support for the concrete language. This is an important point, since building tools by hand is an expensive task. Third, models usually share a common metamodel which directly supports interoperability between different tools that are based on the metamodelling technology at hand. We decided to use Eclipse and the Eclipse Modelling Framework (EMF)[1] because of the variety of tools that exist to create, edit, and transform models based on EMF.

3.1 Language Family Development Process

The initial starting point for developing the ontology-language product line is modelling the problem space by means of a feature model (cf. Fig. 1). We used a lightweight version of the EMF/Ecore-based feature metamodel developed in the feasiPLe project[2].

[1] `http://www.eclipse.org/modeling/emf/`
[2] `http://feasiple.de`

Since we want to (1) transform the description of the solution space, that is, the realisation of the language syntax features and (2) generate tooling (e.g., parsers, printers, editors, ...) out of the specification, we used Ecore to build an OWL metamodel and EMFText[3] [11]—a model-based tool for defining textual concrete syntax for models. EMFText offers a dedicated language for specifying text syntax for models called CS which is similar to Extended Backus-Naur Form (EBNF). With CS, rules are defined which specify textual syntax for metaclasses of a given Ecore-based metamodel. In our case, we first modelled the OWL metamodel (based on [3]) which defines the abstract syntax of the various language features. Then we derived CS rules that describe the textual syntax for the concepts corresponding to OWL Manchester syntax [14].

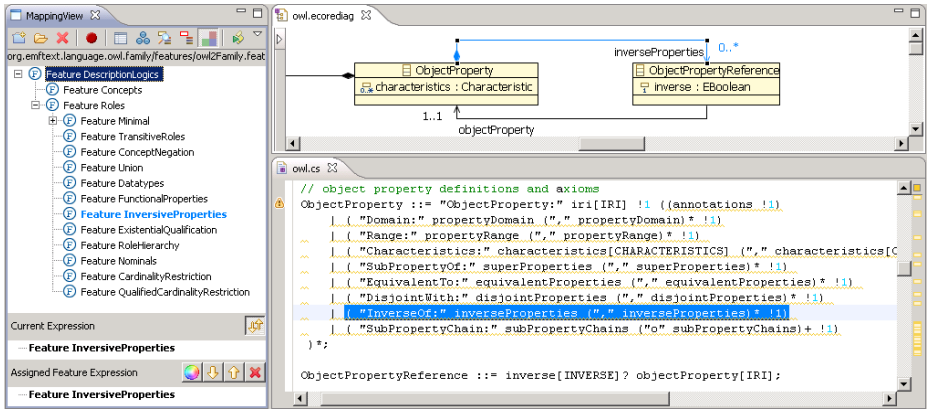

Fig. 2. Using the FeatureMapper to map OWL features to specific parts of OWL2 abstract and concrete syntax

One observation we made in previous work [10, 12] is that it is crucial to have a mapping between the problem space (i.e., a variability description of language features in a feature model) and the solution space (i.e., a concrete realisation of specific language features in EMFText's CS language). This mapping can then be used both for visualising dependencies between features and realisation artefacts and for automating the product-instantiation process. An overview of the models used to specify the the ontology language product line and their relationships is depicted on the right part of Fig 3.

The FeatureMapper[4] is a tool that was specifically developed to tackle that problem and allows for creating a mapping between feature models and EMF-/Ecore-based models. Since EMFText is built with itself, the CS language is again a model-based language which can be used in combination with FeatureMapper. We extended the FeatureMapper to also support mapping and

[3] `http://www.emftext.org`
[4] `http://www.featuremapper.org`

visualisation of textual languages that are created by EMFText and used it to create a mapping between the language features in the feature model and the realisation of those features in the CS specification. Both tools are depicted in Fig. 2 where the view on the left side contains the FeatureMapper with the feature model for the ontology-language product line. The editors on the right side show a part of the OWL metamodel and CS specification for existing languages in the OWL language family. To present the concrete mapping to the language developer the FeatureMapper colours the elements in the CS specification and the OWL metamodel in accordance to the colour of the feature in the feature model they are mapped to. The example depicts the mapping used to define the syntactic realisation of the feature `InverseProperties`. For that purpose the association `inverseProperties` between `ObjectProperty` and `ObjectPropertyReference` that is used in the abstract syntax to represent inverse properties and the corresponding fragment of concrete syntax are mapped to the feature `InverseProperties`.

3.2 Language Derivation Process

After we defined the scope of the language product line, we are able to derive concrete instances (i.e., languages) from that definition. To do so, the FeatureMapper provides means to transform models according to a given feature selection (a variant model) by interpreting the mapping model that contains the various mappings between features and model elements. After this transformation step, a reduced CS specification is produced that only contains the rules that are needed for the selected language features. This reduced CS specification is then used by EMFText to generate a dedicated parser, printer, and editor. The process of feature-based language derivation is depicted in Fig. 3.

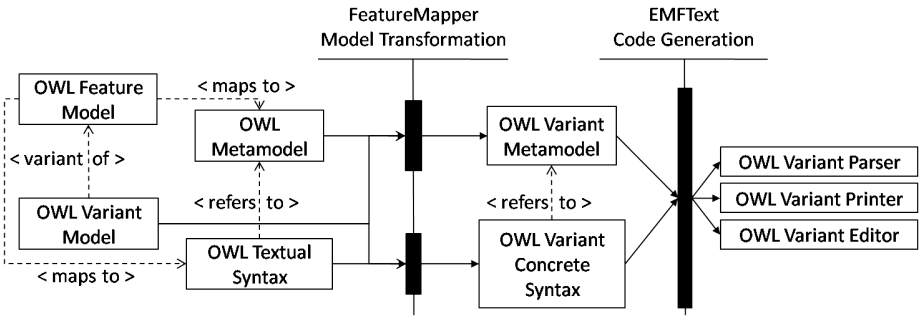

Fig. 3. Model-based Ontology Language Derivation Process

4 Related Work

Scalability is a widely discussed and very prominent topic that constitutes a main challenge for the application of ontology languages [6, 19]. The objective of efficient reasoning has led to a manifold of languages with specific reasoning characteristics [5, 2, 25, 20, 22]. The idea that scalability can be achieved by selecting a language from this manifold that is appropriate for the requirements of a specific use case is not new. In [17] the authors provide a comprehensive comparison of nine DLs-based ontology languages w.r.t. their syntactic features and reasoning efficiency. The results of this survey are envisaged as guideline for matching ontology languages to use cases. The fact that the OWL2 standard [25] introduces three languages with different expressiveness and reasoning characteristics illustrates that nowadays ontology languages are already designed with that idea in mind. Our work picks up this idea and presents a methodical approach and a technological infrastructure to get from language features selected for a use case to the actual implementation of the language variant and the corresponding tool infrastructure. In addition, the presented model-driven approach is suited to deal with the proceeding evolution of ontology languages by supporting the introduction of new language features.

A second branch of work addressing the scalability issue deals with the development of more efficient reasoners or the enhancement of existing reasoning techniques. This led to numerous highly optimised native ontology reasoners [9, 26, 28] that perform well even for expressive ontology languages but only for reasonable sized ontologies. Large amounts of facts often result in poor response times that impede applicability in practice [17]. Approaches presented in [4, 7, 29] store ontologies in relational databases to use the optimised database query engines for ontology reasoning. As discussed in [19] this leads to increased load-time but more efficient reasoning compared to native ontology systems. In [16] ontologies are represented in disjunctive datalog programs. Additional algorithmic optimisation can be applied on the datalog facts to enhance reasoning efficiency. Other approaches [8, 13, 23, 24] enhance reasoning efficiency by approximating more expressive ontology languages to less expressive ones. The reduction of complexity leads to better reasoning performance while preserving the completeness and soundness of the reasoning results. Reasoners and approximation approaches are designed for a very a specific subset of DLs features. Using a generic tool like Sesame [4] that allows for exchanging the reasoning back-end they can be combined with our feature-based ontology language configuration. Thus, we could provide appropriate (semantic) reasoning infrastructure w.r.t. a specific language variant.

5 Conclusion

The contribution of this paper is twofold: First, we transferred the existing DL-based classification of ontology languages to the paradigm of feature modelling. This enables feature-based customisation of OWL2 w.r.t. a specific use case.

Future work needs to enrich the current classification with further metadata (e.g., efficiency annotations) that can be used to guide language configuration. In addition to that, other ontology language extensions—like rule extensions [15] or probabilistic extensions [18]—should be included in such classification.

Second, we presented a model-driven process to generate a dedicated parser, printer, and editor from a given variant specification. This infrastructure is crucial for the application of the language variant in knowledge modelling. In addition, the effort to provide new language extensions or language adaptations can be reduced by using the introduced model-driven process.

The solution presented in this paper only tackles syntactic language variations. To advance the impact on reasoning efficiency and language applicability, future work needs to investigate the possibilities of deriving language specific infrastructure w.r.t. language semantics. This relates to topics as semantic approximation of OWL [24], and composition of language semantics [30].

The introduced approach for applying techniques of product-line engineering for the systematic development of language families is not limited to ontology languages. Thus, future work will also address its extension to other languages.

References

1. F. Baader. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
2. D. Berardi, A. Cali, D. Calvanese, and G. D. Giacomo. Reasoning on UML Class Diagrams. *Artificial Intelligence*, 168, 2003.
3. S. Brockmans, P. Haase, and B. Motik. OWL 2 Web Ontology Language MOF-Based Metamodel. Available at http://www.w3.org/2007/OWL/wiki/MOF-Based_Metamodel, 2007.
4. J. Broekstra, A. Kampman, and F. Van Harmelen. Sesame: A generic architecture for storing and querying rdf and rdf schema. *The Semantic Web ISWC 2002*, pages 54–68, 2002.
5. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *Journal of Automated Reasoning*, 39(3):385–429, 2007.
6. F. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. *Information and Computation*, 134(1):1–58, 1997.
7. Q. Fang, Y. Zhao, G. Yang, and W. Zheng. Scalable Distributed Ontology Reasoning Using DHT-Based Partitioning. *Proceedings of the 3rd Asian Semantic Web Conference on The Semantic Web*, pages 91–105, 2008.
8. P. Groot, H. Stuckenschmidt, and H. Wache. Approximating Description Logic Classification for Semantic Web Reasoning. 2005.
9. V. Haarslev and R. Moller. RACER system description. *Automated Reasoning - Lecture Notes in Computer Science*, pages 701–706, 2001.
10. F. Heidenreich, I. Şavga, and C. Wende. On Controlled Visualisations in Software Product Line Engineering. In *Proceedings of the 2nd International Workshop on Visualisation in Software Product Line Engineering (ViSPLE 2008), collocated with the 12th International Software Product Line Conference (SPLC 2008)*, Sept. 2008.

11. F. Heidenreich, J. Johannes, S. Karol, M. Seifert, and C. Wende. Derivation and Refinement of Textual Syntax for Models. In *Proceedings of the 5th European Conference on Model-Driven Architecture Foundations and Applications (ECMDA-FA 2009)*, June 2009. To appear.
12. F. Heidenreich, J. Kopcsek, and C. Wende. FeatureMapper: Mapping Features to Models. In *Companion Proceedings of the 30th International Conference on Software Engineering (ICSE'08)*, pages 943–944, New York, NY, USA, May 2008. ACM.
13. P. Hitzler and D. Vrandecic. Resolution-based Approximate reasoning for OWL DL. *The Semantic Web ISWC 2005*, 3729, 2005.
14. M. Horridge and P. F. Patel-Schneider. OWL 2 Web Ontology Language: Manchester Syntax. Available at http://www.w3.org/TR/2008/WD-owl2-manchester-syntax-20081202/, 2008.
15. I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. *W3C Member Submission*, 2004.
16. U. Hustadt, B. Motik, and U. Sattler. Reducing SHIQ- description logic to disjunctive datalog programs. *Proceedings of Principles of Knowledge Representation and Reasoning*, pages 152–162, 2004.
17. C. Keet and M. Rodriguez. Comprehensiveness versus Scalability: Guidelines for choosing an appropriate knowledge representation language for bio-ontologies. *KRDB Research Centre Technical Report, KRDB07-5*, 2007.
18. T. Lukasiewicz. Probabilistic Deduction with Conditional Constraints over Basic Events. *Journal of Artificial Inteligence Research*, 1999.
19. L. Ma, Y. Yang, Z. Qiu, G. Xie, Y. Pan, and S. Liu. Towards a complete OWL ontology benchmark. *The Semantic Web: Research and Applications*, 2006.
20. D. McGuinness, R. Fikes, J. Hendler, and L. Stein. DAML+ OIL: an ontology language for the Semantic Web. *IEEE Intelligent Systems*, 17(5):72–80, 2002.
21. D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview. Available at http://www.w3.org/TR/owl-features/, 2004.
22. J. Pan and I. Horrocks. RDFS (FA): connecting RDF (S) and OWL DL. *IEEE Transactions on Knowledge and Data Engineering*, 19:192.
23. J. Pan and E. Thomas. Approximating OWL-DL Ontologies. Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI-07), 2007.
24. J. Pan, E. Thomas, D., and Sleeman. Ontosearch2: Searching and querying web ontologies. *Proceedings of WWW/Internet*, 2006.
25. P. F. Patel-Schneider, P. P. Hayes, and I. Horrocks. OWL 2 Web Ontology Language: Profiles: OWL-R. W3C Working Draft. Available at http://www.w3.org/TR/owl2-profiles/♯OWL-R, 2008.
26. E. Sirin, B. Parsia, B. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical owl-dl reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51–53, 2007.
27. H. Stuckenschmidt. Statement of Interest: Towards Ontology Language Customization. *Ontologies and Information Sharing*, 2001.
28. D. Tsarkov and I. Horrocks. FaCT++ description logic reasoner: System description. *Automated Reasoning - Lecture Notes in Computer Science*, 4130:292, 2006.
29. J. Zhou, L. Ma, Q. Liu, L. Zhang, Y. Yu, and Y. Pan. Minerva: A scalable OWL ontology storage and inference system. *The Semantic Web ASWC 2006*, 4185:429, 2006.
30. S. Zschaler and C. Wende. Collaborating Languages and Tools - A Study in Feasibility. *Technical Report TU Dresden - TUD-FI08-06*, 2008.