# Ontology Engineering Based on Domain Specific Languages and the Application of Ontology Design Patterns

Thomas Janke

SAP Research Dresden, Germany
thomas.janke@sap.com

**Abstract.** Knowledge management is a key success factor for companies. It can be facilitated by building information systems based on semantic technologies and herewith gain the profits of describing knowledge in a machine processable and exchangeable way. The problem companies are facing today is that ontology engineering still is a very complex and challenging task. Modeling high quality ontologies, in the sense of using the best suited language for the domain of interest with respect to available modeling constructs as well as applying established modeling best practices, is very difficult. This is especially true for domain experts who normally have no or just limited knowledge about the underlying logical formalisms. The model driven ontology engineering platform presented in this paper tackles those problem in two ways: 1) Domain specific languages (DSL) are provided which abstract from concrete ontology languages and hereby simplify ontology modeling for domain experts. 2) Ontology design patterns (ODP)[7], which describe best practices for ontology engineering, are applied automatically and transparently during ontology generation. In order to examine the feasibility of this approach, a prototype has been developed on top of the Eclipse Modeling Project (EMP).

**Keywords:** Domain Specific Languages, Ontology Engineering, Domain Specific Knowledge Engineering, Ontology Design Patterns.

## 1 Introduction

Ontology engineering is hard to master. The reasons for that are manifold and cover methodical as well as technological issues. The ontology engineering platform presented in this paper has been designed and implemented based on the assumption that the most severe problem comes with the fact that ontology engineers need to have a well-founded knowledge about the underlying logical formalisms (frames, first order logic, description logics, semantic networks, etc.) in order to model high-quality ontologies. Important aspects are the selection of the most appropriate language for a given scenario from the pool of available ontology languages (RDF(S), OWL, Topic Maps, F-Logic, etc.) as well as the knowledge about where to find and how to apply established modeling best practices. Typically, this cannot be assumed to be true or is at least difficult for people that do not have a grounded logical background. This fact stands in a vast contrast to the idea that domain experts should be capable of describing their field of expertise by using ontologies as their modeling language[7]. Today, this gap can only be filled by an intense collaboration of domain experts and ontology engineers, whereas the latter first need to understand the experts knowledge, to be able to then express this as formal descriptions. This approach leads to high costs in terms of both: development time and money to buy the expertise needed for ontology engineering.

To overcome or at least mitigate this situation, an ontology engineering platform, based on domain specific languages, the transparent application of ODPs and the concepts of model driven development (MDD), is proposed in this paper. It enables domain experts to model their domain of expertise using a domain specific language instead of general purpose ontology languages. The provided DSL abstracts from concrete ontology languages and incorporates terms and concepts the domain expert is familiar with. Based on this description, concrete ontologies in various languages can be generated. The transformations used to create those ontologies are provided by ontology experts. Therefore, they are designed in a way, that well known and documented modeling best practices, so called ontology design patterns, are reflected properly. The main benefits of such an approach are twofold. On the one hand it simplifies the modeling of ontologies for domain experts and on the other hand, by automatically and transparently applying best practices, it leads to higher modeling quality.

The paper is structured in the following way: Related research fields are introduced and preliminary contributions are refined from the envisioned platform. Thereafter, the architecture of the approach is described in more detail, the applied meta models are introduced, the developed domain specific language is discussed and a model editor is presented. Finally, the contributions are summarized, results are discussed and an outlook on future work is given.

## 2 Related Work

The combination of domain specific languages and ontologies has been discussed in various publications, for example [13] and [14]. The main focus of this research lies on how to improve the engineering of DSLs with the help of ontologies, facilitating their formal semantics. In contrast, the approach described in this paper uses domain specific languages to abstract from concrete ontology languages in order to simplify ontology engineering and improve ontology quality by transparently applying ontology design patterns.

ODPs are, in analogy to software design patterns, reusable, successful solutions to recurrent modeling problems[4]. The application of design patterns in the field of ontology engineering has been discussed in many publications[10,12,6,8,5] and attracts a growing community[2]. ODPs are also addressed by a dedicated W3C working group[3]. Following the classification of patterns given in [7], the platform introduced in this paper addresses so called *Transformational*

*Ontology Patterns*. They translate expressions from one language into another, trying to approximate the semantics of the source language. An example for such a pattern is the handling of n-ary relations, which are not supported as first level language primitives in OWL and RDF(S). Both of them only support binary relations. Nevertheless, there are published best practices how to model n-ary relations, for example by the W3C[1]. Another example of a design pattern is the description on how to model inverse relations. They are supported as default language constructs by OWL but not, for example, by F-Logic. However, there are best practices how to model inverse relations in F-Logic too. Whereas those patterns are a good starting point for optimizing ontology modeling by applying modeling best practices, they still put certain demands on domain experts. First of all, they need to be aware of ODPs, then they have to find the adequate pattern for a special task as well as understand its semantic and finally they have to apply the pattern manually. The platform presented in this paper offers authors of ODPs the possibility to provide their patterns not only in text form but also in an executable way.

## 3 Architecture of the Ontology Engineering Platform

This paper introduces an ontology engineering platform based on textual domain specific languages and ontology generation on top of ontology design patterns. The overall architecture of the platform is illustrated in Fig. 1. Ontologies are modeled with the help of domain specific languages. Those languages offer a textual syntax, containing keywords and concepts well understood in the particular domain they are designed for. The domain specific descriptions, in turn, are mapped to a common ontology model. Based on this model, concrete ontologies can be generated. Due to the fact that the platform is designed based on concepts and frameworks for model driven development, this is realized by means of model to model transformations. The latter are defined based on the common ontology meta model and the meta models of concrete ontology languages. In the architecture presented in Fig. 1, the OWL meta model is listed as an example. Each transformation is reflected by a collection of ontology design patterns. Those patterns are modular in the sense that each and every pattern describes a distinct and self-contained modeling best practice. Examples for patterns are n-ary relations, inverse relations, transitive relations, cardinality constraints for properties and transitive reduction. In order to apply a pattern to the transformation process and herewith execute it during ontology generation, the pattern has to be registered to the transformation runtime.

The architecture has been implemented based on components developed in the Eclipse Modeling Project (EMP) and its sub projects. Hence, Ecore has been used to define the common ontology meta model depicted in Fig. 1. Furthermore, EMF tooling has been used to develop a modular model transformation framework, which is the main component in the ontology generation module. In addition, the Xtext[1] framework has been used to develop a domain specific

[1] http://www.eclipse.org/Xtext/

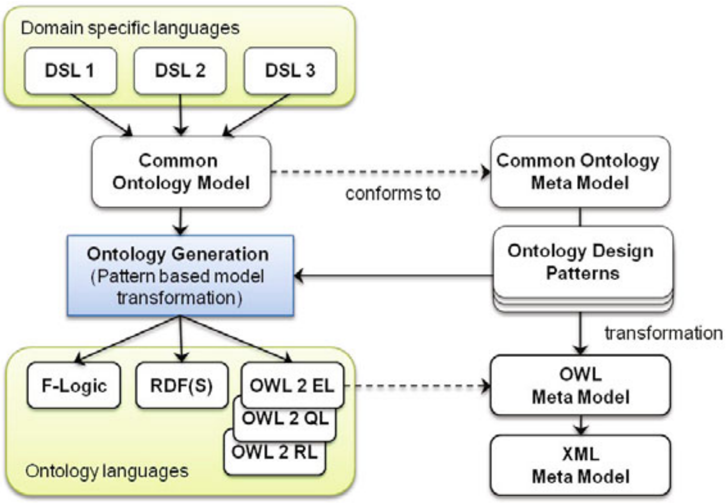


**Fig. 1.** A model driven ontology engineering platform based on ODPs

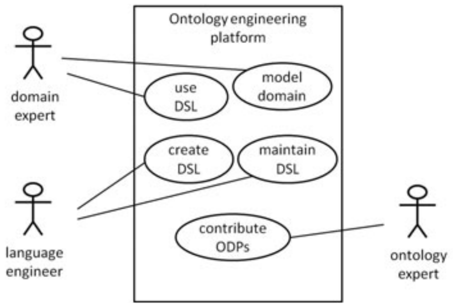


**Fig. 2.** Actors and use cases involved in the ontology engineering process

language for creating common ontology models and to generate an Eclipse editor to support domain modeling by offering syntax checking and code completion.

As depicted in Fig. 2, the platform allows for separating concerns along the competencies of all stakeholders involved in the modeling process. The domain expert is responsible for modeling his domain by using the provided domain specific language. The language engineer, in turn, concentrates on designing and maintaining the DSL with respect to the requirements of the given domain. In order to do that, he must not be an ontology engineer but only understand the common ontology model. Apart from that, he still needs to do a requirements analysis together with the domain expert in order to design a suitable language. Nevertheless, after this initial effort, domain experts can model and maintain their domain without further assistance. The third stakeholder involved are ontology experts which, using the interface given by the common ontology model, can contribute their knowledge by means of ODPs.

## 4 Applied Ontology Meta Models

The prototype is based on 4 meta models: meta models for RDF(S) and OWL as well as a XML meta model and the common ontology meta model developed as part of the prototype.

The XML meta model is provided by the AMMA[2] project as the standard meta model used to read and write XML models. The two ontology meta models are part of an Atlas Transformation Language (ATL) use case[3] published by the SIDo Group of the L3I[4] showing how to transform UML models into ontologies.

The common ontology meta model is used as a pivot language to describe the domain of interest and therefore is used as the starting point for model transformations to the concrete ontology meta models. As a result, all ontology design pattern implementations describe transformations from elements of this model to elements of the OWL and RDF(S) meta model, respectively. Therefore, every domain specific language developed in the context of this platform is free to reflect as many language features provided by the common meta model as needed. Whereas some domains may only require a small subset of language construct, others may need all of them to be provided.

The meta model developed in order to show the feasibility of the approach supports the following modeling primitives: *entities* and *entity hierarchies*, *attributes*, *relations* and *relation hierarchies*, *inverse relations*, *transitive relations*, *functional relations*, *n-ary relations* and *relation cardinalities*. Moreover, *namespaces* and *imports of other models* can be expressed. Although this collection of primitives reflects only a subset of features provided by common ontology languages, its expressivity is sufficient to model widely spread and used ontologies like Friend of a Friend (FOAF) or Dublin Core (DC). Moreover, it is adequate to apply a number of patterns for various languages as a proof of concept for the approach. In later iterations, it needs to be evaluated how to add more complex primitives like rules, annotations for instances and other axioms.

Apart from that, it is important to mention that this abstraction comes at a price: first, the common model will not be as expressive as concrete ontology languages, and second, the transformation from the common model to a concrete ontology model can only *approximate* the semantics of the applied primitive[7]. Nevertheless, the existence and the spread of such patterns[2,3,11,9] shows that this fact is acceptable in many real world scenarios and applications.

## 5 Design of Domain Specific Languages

In order to provide appropriate tool support for both - language engineering and domain modeling - Xtext has been used in the prototype to specify an easy to use ontology DSL. The Xtext framework supports the definition of languages and, beyond that, offers tooling to generate the whole language infrastructure,

[2] AtlanMod Model Management Architecture.
[3] `http://www.eclipse.org/m2m/atl/usecases/ODMImplementation/#download`
[4] `http://l3i.univ-larochelle.fr/`

including parser implementations as well as Eclipse-based editors with code completion and syntax checking. The concrete syntax for the developed language is specified using the Xtext grammar language. The abstract syntax is given by means of the meta models that are going to be populated by an automatically generated parser. Specifically, this means that ontology models, described in the developed language, are parsed and corresponding models, which conform to the common ontology meta model, introduced in Sect. 4, are instantiated. Using this approach, multiple DSLs can be specified, all based on one common ontology meta model. Due to the fact that Xtext is based on EMF, besides textual languages, also editors supporting graphical notations can easily be derived, for example by applying the Graphical Modeling Framework (GMF).

As outlined in Fig. 2, a domain specific language needs to be designed by a language engineer reflecting the requirements of the given domain. Therefore, an analysis must, among others, answer the following questions:

- What primitives, for example transitivity or constraints for relations, are needed in order to describe the domain?
- Are there any models that need to be reused or imported?
- What are common queries to be answered with the help of the ontology?

At the time of writing, we conduct a requirement analysis in the domain of software quality engineering for embedded devices. The results will be used to develop a methodology for building DSLs on top of the common ontology model for various domains. Additionally, the outlined approach can be further evaluated based on the experiences gained in this domain.

The language developed as part of the prototype, in its early state, is not yet bound to a specific domain but rather offers keywords still close to common ontology concepts. As a consequence, there are keywords like *ontology*, *concept* and *property*. Nevertheless, it illustrates already, how certain ontology primitives can be encapsulated and shielded from the domain expert by means of abstraction. Listing 1.1 contains a simplified example of the current modeling language.

```
ontology 'http://demo' {
   //import owl ontology
   import "../ontologies/foaf.owl"
   // import another domain model
   import "../ontologies/bookstore.oml"

   concept Customer {
      property name : string
      property account : "http://xmlns.com/foaf/0.1/#OnlineAccount"
      // n-ary relation
      property bought contained as 'Purchase' : Book as 'item', Store
          [1,1]
   }
}
```

**Listing 1.1.** Simplified ontology model

At first, an ontology is defined given a unique name. An obvious pattern implementation takes this name and uses it as the base URI of the ontology. Subsequently, two imports are defined: the first one is referencing an OWL ontology, namely FOAF, and the second one refers to another domain model described using the same domain specific language. Imports are used to enable and foster reuse of existing ontologies, which is a key concept of ontologies with respect to their goal to provide shared vocabularies. As shown here, imports across different ontology languages are possible. The current prototype supports imports by means of exposing all concepts defined in the referenced ontologies to the importing model. In the given example, this is illustrated by the *account* property, which is defined for the concept *Customer*. The property references the concept *OnlineAccount*, which is not part of the actual ontology but is imported from the FOAF ontology as its range. The same applies for the concepts *Book* and *Store* which both are defined in the cross referenced *bookstore* ontology. The developed editor supports those kind of imports by offering full auto completion for concept names. The patterns responsible for mapping the concept of imports to the corresponding RDF(S) and OWL models are very similar. Both of them generate a property range containing the fully qualified concept URI of the cross referenced concept. In addition, the OWL pattern generates an import statement making use of the `owl:import` property. The latter is not available in RDF(S) and is therefore omitted in those ontologies.

The last property defined in the ontology is an example of a n-ary relation which can be expressed in two different ways: Either a normal property definition is extended by the `contained` keyword or more than one concept and simple data type, respectively, are defined as the property range. Additionally, a cardinality constraint is posed on *Store* using the common min-max notation. Given that, the *bought* property is defined as an n:n:1 relation. Fig. 3 illustrates one of the patterns implemented to realize n-ary relations in the prototype. It is based on a best practice recommended by the W3C[1]. In order to model the n-ary relation *bought*, a new "link" concept named *Purchase* is introduced, which takes the role of the respective relation. The specific name has been specified with the help of the `as` keyword. If this would not have been the case, a name would have been automatically generated. The link concept is used as a domain for three newly introduced properties, namely *item*, *has_Store* and *has_Customer*. Whereas the last two have generated names, the *item* property has been labeled explicitly,

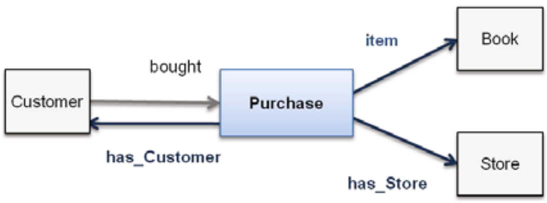


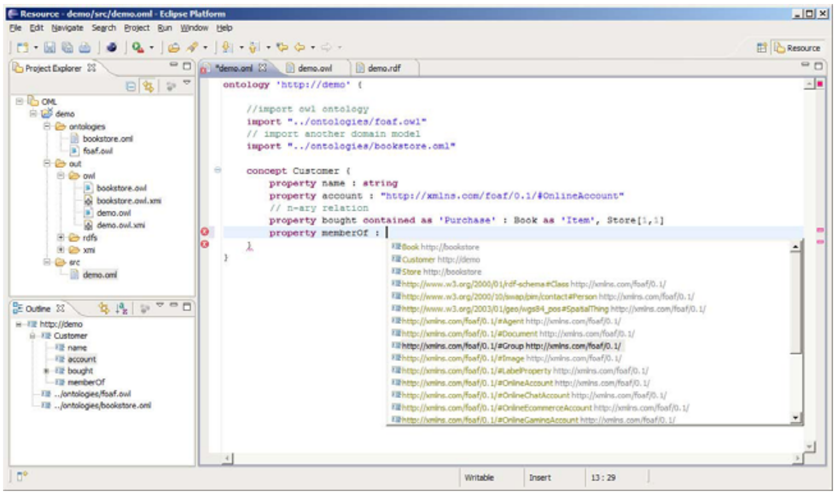
**Fig. 3.** Pattern for n-ary relations

again using the `as` keyword. For languages that support inverse relations, the *bought* and the *has_Customer* relations are additionally marked as inverse.

The cardinality constraint on *Store* can only be reflected in OWL ontologies, while RDF(S) does not support such restrictions. Therefore, two different patterns have to be applied. In OWL, a constraint on *has_Store* is generated, using `owl:cardinality`. For RDF(S), missing constraints are documented in the ontology by means of comments. Those annotations may later be used to implement the missing functionality on top of the ontology in the business logic layer of the targeted application. Cardinality constraints posed on non n-ary relations are handled in the same way. In OWL, `owl:minCardinality` and `owl:maxCardinality` are used to reflect those constraints in a concrete ontology.

## 6 A Domain Specific Ontology Editor

In addition to the model transformation framework and the common ontology meta model, an editor (see Fig. 4) is provided which supports the development of domain models using a textual domain specific language. The corresponding Eclipse plugin is based on an editor generated by the Xtext framework for the grammar of a given DSL. On top of that, the editor has been extended by various features, like code completion support for concepts imported from OWL and RDF(S) ontology files.



```
ontology 'http://demo' {

    //import owl ontology
    import "../ontologies/foaf.owl"
    // import another domain model
    import "../ontologies/bookstore.oml"

    concept Customer {
        property name : string
        property account : "http://xmlns.com/foaf/0.1/#OnlineAccount"
        // n-ary relation
        property bought contained as 'Purchase' : Book as 'Item', Store[1,1]
        property memberOf : 
    }
}
```

```
Book http://bookstore
Customer http://demo
Store http://bookstore
http://www.w3.org/2000/01/rdf-schema#Class http://xmlns.com/foaf/0.1/
http://www.w3.org/2000/10/swap/pim/contact#Person http://xmlns.com/foaf/0.1/
http://www.w3.org/2003/01/geo/wgs84_pos#SpatialThing http://xmlns.com/foaf/0.1/
http://xmlns.com/foaf/0.1/#Agent http://xmlns.com/foaf/0.1/
http://xmlns.com/foaf/0.1/#Document http://xmlns.com/foaf/0.1/
http://xmlns.com/foaf/0.1/#Group http://xmlns.com/foaf/0.1/
http://xmlns.com/foaf/0.1/#Image http://xmlns.com/foaf/0.1/
http://xmlns.com/foaf/0.1/#LabelProperty http://xmlns.com/foaf/0.1/
http://xmlns.com/foaf/0.1/#OnlineAccount http://xmlns.com/foaf/0.1/
http://xmlns.com/foaf/0.1/#OnlineChatAccount http://xmlns.com/foaf/0.1/
http://xmlns.com/foaf/0.1/#OnlineEcommerceAccount http://xmlns.com/foaf/0.1/
http://xmlns.com/foaf/0.1/#OnlineGamingAccount http://xmlns.com/foaf/0.1/
```

**Fig. 4.** A domain specific ontology editor

In order to seamlessly integrate the ontology generation process, project builders for OWL and RDF(S) ontologies are provided additionally. They are responsible for transforming the model described in the editor into its corresponding ontology models and are triggered by the model build process. As a result, corresponding OWL and RDF(S) ontologies are generated automatically and stored in XML format whenever the model has been modified, ensuring the liveness of the models during their maintenance. The modeling as well as the transformation process is assisted by markers for errors and warnings. Furthermore, wizards for creating new modeling projects as well as domain model files are provided. Based on this tool chain, domain experts are enabled to model their domain of interest in a domain specific language getting ontologies, modeled by applying well established best practices, in various languages as a result.

## 7 Summary and Future Work

Modeling high-quality ontologies is very challenging, especially for non-expert without a grounded knowledge about the underlying logical formalisms. This applies to both, selecting the most suitable ontology language and applying modeling best practices. In order to tackle those problems, an ontology engineering platform based on domain specific languages and ontology design patterns has been introduced. It aims at simplifying the ontology modeling process for domain experts by providing domain specific modeling languages which abstract from concrete ontology languages. Based on that abstraction, provided by means of a common ontology meta model, ontologies in various languages can be generated from the very same domain description. Moreover, ontology design patterns, which are published and widely accepted modeling best practices, are transparently and automatically applied during the ontology generation process, helping to increases the modeling quality of the emitted ontologies. In order to show the feasibility of the outlined approach, a prototype has been developed which covers the most important aspects and components required for a fully functional modeling platform. This also includes the design and implementation of a modular and extensible model to model transformation framework.

Whereas the experiences gained while designing and implementing the outlined prototype are promising, the suitability of the proposed platform can only be validated if the common ontology model and the DSLs provided have a appropriate expressivity. The ontology design patterns introduced in this paper and implemented as part of the prototype are a good starting point in order to show the benefits of the platform and to trigger further discussions and development. Further patterns and ontology model extension need to be developed on top of the provided platform. Suitable candidates for the ongoing development are annotations for triples (e.g., via reification), rules and the import of properties defined in external ontologies.

Furthermore, the modeling language introduced is this paper has to be extended to better reflect the specifics of a certain domain. In order to achieve that and to evaluate the platform in a real life use case, we started the development of such a language in the domain of ambient assisted living. The workbench will be used in order to model the interfaces as well as data quality aspects of embedded devices. The results of this work will help to compile additional requirements and to review the proposed approach.

Another aspect that needs to be investigated with regard to the real life applicability of the platform is how to sufficiently support editing of generated ontologies. This is a rather general problem of model driven approaches. In this particular domain it means that manual changes should not be overwritten during regeneration of the corresponding ontology. As a result, the ontology generation component has to be aware of manual changes. Proposals for potential solution can probably be derived from the implementation of the EMF code generation framework.

## References

1. Defining N-ary Relations on the Semantic Web, `http://www.w3.org/TR/swbp-n-aryRelations/`
2. Ontology Design Patterns, `http://ontologydesignpatterns.org`
3. Semantic Web Best Practices and Deployment Working Group (2001), `http://www.w3.org/2001/sw/BestPractices/`
4. Blomqvist, E.: D2.5.2: Pattern based ontology design: methodology and software support, NeOn project (2010)
5. Blomqvist, E.: Ontology patterns: Typology and experiences from design pattern development. In: Linköping Electronic Conference Proceedings, pp. 55–64. Linköping University Electronic Press, Linköpings universitet (2010)
6. Blomqvist, E., Sandkuhl, K.: Patterns in ontology engineering: Classification of ontology patterns. In: ICEIS, vol. 3, pp. 413–416 (2005)
7. Gangemi, A., Presutti, V.: Ontology Design Patterns ODP. International Handbooks on Information Systems. In: Handbook on Ontologies, Springer, Heidelberg (2009)
8. Presutti, V., Gangemi, A.: Content ontology design patterns as practical building blocks for web ontologies. In: Li, Q., Spaccapietra, S., Yu, E., Olivé, A. (eds.) ER 2008. LNCS, vol. 5231, pp. 128–141. Springer, Heidelberg (2008)
9. Presutti, V., Gangemi, A., David, S., Aguado de Cea, G., Suarez Figueroa, M.-C., Montiel-Ponsoda, E., Poveda, M.: D2.5.1: Library of design patterns for collaborative development of networked ontologies, NeOn project (2008)
10. Staab, S., Erdmann, M., Maedche, A.: Engineering ontologies using semantic patterns. Ontologies and Information Sharing (2001)
11. Suarez Figueroa, M.-C., Brockmans, S., Gangemi, A., Gómez-Pérez, A., Lehmann, J., Lewen, H., Presutti, V., Sabou, M.: D5.1.1: NeOn Modelling Components, NeOn project (2007)
12. Svátek, V.: Design patterns for semantic web ontologies: Motivation and discussion. In: 7th Conference on Business Information Systems, Poznan (2004)
13. Tairas, R., Mernik, M., Gray, J.: Models in software engineering. In: Chaudron, M.R. (ed.) Using Ontology in the Domain Analysis of Domain-Specific Languages, pp. 332–342. Springer, Heidelberg (2009) ISBN: 978-3-642-01647-9
14. Walter, T., Silva Parreiras, F., Staab, S.: OntoDSL: An ontology-based framework for domain-specific languages. In: Schürr, A., Selic, B. (eds.) MODELS 2009. LNCS, vol. 5795, pp. 408–422. Springer, Heidelberg (2009)