WP1-D4

Revision of the combined metamodel

Project title:	Marrying Ontology and Software Technology
Project acronym:	MOST
Project number:	ICT-2008-216691
Project instrument:	EU FP7 STREP
Document type:	D (deliverable)
Nature of document:	R (report)
Dissemination level:	PU (public)
Document number:	ICT216691/UoKL/WP1-D4/D/PU/a1
Responsible editors:	Tobias Walter, Fernando Silva Parreiras, Gerd Gröner
Reviewers:	Mirko Seifert, Yuting Zhao
Contributing participants:	UoKL
Contributing workpackages:	WP1
Contractual date of deliverable:	31 July 2010
Actual submission date:	17 July 2010

Abstract
To enable ontologies to leverage model-driven software development and other software processes, a conceptual integration of the ontology and the model-driven software development paradigm must take place. To achieve such an integration, the deliverable fulfills the requirement of an integrated metalanguage on level M3, as well as consistently defined, integrating metamodels on level M2 to connect ontologies to the model-driven software development model chain. Based on experiences collected with the initial prototype the joint metamodel, its querying and its transformation, we revise the patterns identified in previous deliverables. In contrast to our first deliverable D1.1 we mainly concentrate on the user scenarios and the languages which are used in the case studies.

Keyword List
software modelling, ontologies, integration, metamodel

Project co-funded by the European Commission and the Swiss Federal Office for Education and Science within the Seventh Framework Programme.

Revision of the combined metamodel

Tobias Walter[1], **Fernando Silva Parreiras**[1], **Gerd Gröner**[1]

[1] Institute for Web Science and Technology
University of Koblenz-Landau
Email: {walter, parreiras, groener }@uni-koblenz.de,

17 July 2010

Abstract
To enable ontologies to leverage model-driven software development and other software processes, a conceptual integration of the ontology and the model-driven software development paradigm must take place. To achieve such an integration, the deliverable fulfills the requirement of an integrated metalanguage on level M3, as well as consistently defined, integrating metamodels on level M2 to connect ontologies to the model-driven software development model chain. Based on experiences collected with the initial prototype the joint metamodel, its querying and its transformation, we revise the patterns identified in previous deliverables. In contrast to our first deliverable D1.1 we mainly concentrate on the user scenarios and the languages which are used in the case studies.

Keyword List
software modelling, ontologies, integration, metamodel

Contents

Change Log

Version	Date	Author(s)	Changes
1.0	31.03.10	Tobias Walter	Document created
1.1	31.05.10	Tobias Walter	Integration Bridges created
1.2	31.07.10	Tobias Walter	Transformation Bridges added
1.3	30.08.10	Fernando Parreiras	TwoUse Toolkit described
1.4	15.08.10	Tobias Walter	Internal review comments instructed

List of Figures

List of Tables

Terms and Definitions

Abstract Syntax. The abstract syntax delineates the body of concepts and how they may be combined to create models. It comprises definitions of the concepts, the relationships between concepts and well-formedness rules.

Concrete Syntax. The concrete syntax provides the notation to present the constructs defined in the abstract syntax. It can be categorised in textual syntax and visual syntax.

Class-based modelling. Class-based modelling is an approach consisting basically of the essential constructs used to model a UML class diagram that are common to other metamodelling approaches like MOF and Ecore. Examples of these constructs are Class, Property, Operation, Classifier, Attribute.

UML-based Models. UML-based models are models described by metamodels using different architectures derived or based on UML. Examples of UML-based models are MOF models, Ecore models, SysML models or BPMN models.

Metamodel. Metamodel is a model defined on the M2 level.

Metaclass. Metaclass is the class construct on the M2 level according to the the OMG's Four layered metamodel architecture. In fact, when describing metamodels, metaclasess are simply referred to as classes.

Model Transformation. Model transformation is a function that receives a source model, a source metamodel, a target metamodel and a transformation script as input and produces a target model conforming to a target metamodel.

Metamodelling Architecture. Metamodelling Architecture comprises the set of metamodels and packages declared on M2 level, one or more concrete syntaxes to design models conforming with the set of metamodels and mapping rules to accomplish the translation from the concrete syntax to the abstract syntax (metamodels).

UML-based Metamodelling. UML-based metamodelling consists of different metamodels that use constructs, such as class, property and operation as essential constructs. We use the term UML-based metamodelling to collectively refer to the metamodels UML, MOF and Ecore.

OMG's Four layered metamodel architecture. It is an architecture defined by OMG with four different levels: the metametamodel level (M3), the metamodel level (M2), the model level (M1) and the objects level (M0) (or real world).

Ontology. In this document, the terms ontology and OWL ontology are used interchangeably. For using UML profiles we focus on OWL as language for ontologies.

List of Abbreviations

AS	Abstract Syntax
KB	Knowledge Base
CS	Concrete Syntax
CbML	Class-based modelling language
CWA	Closed World Assumption
DFA	Deterministic Finite Automaton
DL	Description Logics
DSL	Domain Specific Language
M0	Metamodel Level 0
M1	Metamodel Level 1
M2	Metamodel Level 2
M3	Metamodel Level 3
MBOTL	Model-based Ontology Translation Language
MDA	Model Driven Architecture
MM	Metamodel
MOF	Meta-Object Facility 2.0
MOST	Marrying Ontologies and Software Technologies
MTL	Model Transformation Language
NA	Not Available
OCL	UML 2.0 Object Constraint Language
ODM	Ontology Definition Metamodel
OMG	Object Management Group
OWA	Open World Assumption
OWL	Web Ontology Language
OWL 2	Web Ontology Language 2
OWL DL	The Description Logics Dialect of OWL
OWL Full	The Most Expressive Dialect of OWL
QVT	Query / View / Transformation
RDF	Resource Description Framework
RDFS	RDF Schema
UML	Unified Modeling Language 2.0
WFR	Well Formedness Rules

1 Introduction

Today *Model Driven Development* (MDD) plays a key role in describing and building software systems. A variety of different software modeling languages may be used to develop one large software system. Each language focuses on different views and problems of the system [Mellor et al., 2003]. *Model Driven Software Engineering* (MDSE) is related to the design and specification of modeling languages and it is based on the four-layer modeling architecture [Atkinson and Kühne, 2003]. In such a modeling architecture the M0-layer represents the real world objects. Models are defined at the M1-layer, a simplification and abstraction of the M0-layer. Models at the M1-layer are defined using concepts which are described by metamodels at the M2-layer. Each metamodel at the M2-layer determines how expressive its models can be. Analogously metamodels are defined by using concepts described as metametamodels at the M3-layer.

Although the four-layer modeling architecture provides the basis for formally defining software modeling languages we have analyzed some open challenges. Semantics of modeling languages often are not defined explicitly but hidden in modeling tools. To fix a specific formal semantics for metamodels, it should be defined precisely in the metamodel specification. The syntactic correctness of models is often analyzed implicitly in procedural checks of the modeling tools. To make well-formedness constraints more explicit, they should be defined precisely in the metamodel specification.

OWL2, the web ontology language, is a W3C recommendation with a very comprehensive set of constructs for concept definitions [Motik et al., 2009] and constitutes formal models of domains. Since ontology languages are described by metamodels and can be developed in a model-driven manner, they provide the capability to combine them with software modeling languages.

The objective of this deliverable is to revise the initial metamodel based on experiences gained with the first demonstrator. We show how ontologies can support the definition of software modeling language semantics and provide the definition of syntactic constraints. Since OWL2 has not been designed to act as a metamodel for defining modeling languages we propose to build such languages in an integrated manner by bridging pure language metamodels and an OWL metamodel in order to benefit from both.

1.1 Relation to previous deliverables

1.1.1 D1.1 - Report on the combined metamodel

Deliverable 1.1 [Parreiras and Walter, 2008] presents a framework involving the integration of existing metamodels and profiles for UML and OWL modeling, including relevant (sub)standards such as OCL and considering newer developments such as SWRL, a weaving metamodel and an UML profile for developing integrated models. This deliverable reviews the concepts and the framework defined in D1.1 in different metamodeling layers, namely M2 and M3.

1.1.2 D1.2 - Report on querying the combined metamodel

Deliverable 1.2 [Zhao et al., 2009] describes a querying solution to support developers in querying and transforming integrated models.

1.1.3 D1.3 - D1.3 Report on transformation patterns

Deliverable 1.3 [Parreiras et al., 2009] describes frequently used patterns for transforming instantiations of metamodeling technical spaces into intermediate models, code and reasoning access. The main objective is to explore the mediation patterns that are used in order to transform, combine and refine models. This deliverable refines the transformation patterns described in D1.3. The transformation approaches and pattern identified in D1.3 are reused by the M3 transformation bridge (cf. Section 3.2) and the M2 transformation bridge (cf. Section 4.2).

2 Modeling and Metamodeling in MOST

2.1 Use Cases in MOST

In the following we want to refer to the main use cases in MOST which come from the industrial partners, namely *Comarch* for Workpackage 5 and *SAP* for Workpackage 6. Based on the use cases we discuss in Section 3 and 4 the different metamodelling and modelling approaches.

2.1.1 Workpackage 5 Use Cases

In [Wende, 2009] concrete examples and problems in MDE with respect to the case study of Comarch are presented.

Here the need for consistent management of physical network devices is described, where devices can have thousands of different configurations with many different slots and plugged in cards. Figure 1 depicts a model physical device model. The model specifies that the device has three slots and that the first slot is required (marked red in the diagram). The possible or required cards are indicated in blue rectangles next to the respective slots. Having a model of a physical device the following key questions arise:

1. Network planning: what components can I use with a given configuration of a device to build my service?
2. Consistency checks: is my configuration valid? Is my knowledge base about possible configurations consistent?
3. Data quality: what is the exact type of device, given its configuration?
4. Explanations and guidance: Why is the configuration invalid? How can I fix it?

To answer those key questions a first step is to provide models and metamodels with constraints and underlying formal semantics. In Section 3.1 we present approaches for defining constraints and formal semantics by ontology languages.

2.1.2 Workpackage 6 Use Cases

In [Wende, 2009] concrete examples and problems in MDE with respect to the case study of SAP are presented.

SAP is dealing with business process modelling and management. Here, a common task is to refine the process models to represent the design of processes on different levels of abstraction. An abstract process describes the core functionality and behaviour of an application. A

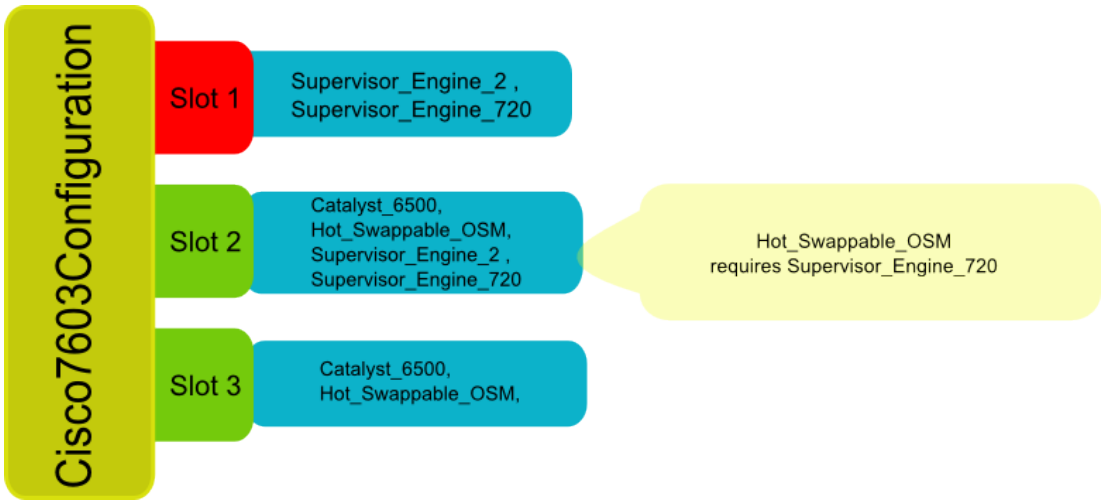

Figure 1: Example physical device model

refinement is an extension of an activity into a more specific process description. In such a procedure, the refined process should refer to the intended behaviour of the abstract process and satisfy behavioural constraints of existing software components. However, due to a number of possibly complex refinement steps, such a relation between the original generic process and the refined specific process is not always obvious. Therefore, to check and ensure the consistency of a refinement becomes a crucial issue in process management.

2.2 Modelling in MOST

A relevant initiative from the software engineering community called Model Driven Engineering (MDE) is being developed in parallel with the Semantic Web [Mellor et al., 2003]. The MDE approach suggests first to develop models describing the system in an abstract way, which later is transformed into real, executable systems (e.g. source code) or ontologies used by reasoners [Parreiras et al., 2009]. The MDE process is embedded in the process of ontology-driven software engineering [Wende, 2009].

To advance understanding and usability of models, models must have a meaning and must conform to a given structure. In MDE models are described by software languages, where software languages themselves are described by so called metamodeling languages. A language consists of an *abstract syntax*, at least one *concrete syntax* and *semantics*.

The abstract syntax of a software language is described by a metamodel and is designed by a *language designer*. Semantics of the language may be defined by a natural language specification or may be captured (partially) by logics. A concrete syntax, which could be of textual or visual kind, is used by a *language user* to create software models.

Since metamodels are also models, metamodeling languages are needed, to describe software languages. Here the abstract syntax is described by a metametamodel. In MOST the following metametamodels are considered.

grUML In the scope of graph-based modeling to create software models [Ebert, 2008] a metamodeling language (e.g. grUML [Bildhauer et al., 2008]) must allow for defining graph

schemas, which provide types for vertices and edges and structures them in hierarchies. Here each graph is an instance of its corresponding *schema*.

MOF The Meta-Object Facility (MOF) is OMG's standard for defining metamodels. It provides a language for defining the abstract syntax of modeling languages. MOF is in general a minimal set of concepts which can be used for defining other modeling languages. The version 2.0 of MOF provides two metametamodels, namely *Essential MOF* (EMOF) and *Complete MOF* (CMOF). EMOF prefers simplicity of implementation before expressiveness. CMOF instead is more expressive, but more complicated to implement [OMG, 2006]. EMOF mainly consists of the Basic package of the Unified Modeling Language (UML) which is part of the UML infrastructure [OMG, 2007a]. It allows for defining classes together with properties, which are used to describe data attributes of classes and which allow for referencing to other classes.

Ecore Another metametamodel is provided by the Ecore metamodeling language, which is used in the Eclipse Modeling Framework [Budinsky et al., 2003]. It is similar to EMOF and will be considered in the rest of this paper. Ecore provides four basic constructs: (1) EClass - used for representing a modeled class. It has a name, zero or more attributes, and zero or more references. (2) EAttribute - used for representing a modeled attribute. Attributes have a name and a type. (3) EReference - used for representing an association between classes. (4) EDataType - used for representing attribute types.

ADOxx The ontology-aware model management in MOST is based on the *ADOxx Metamodelling Environment*. Like other modelling frameworks and environments it is also based on a metametamodel, called *ADOxx Meta*^{2}*-Model*, which is used for the definition of the visual modelling languages within the framework. In general, it contains constructs like classes and associations, which are also available in other metametamodels. A detailed definition of the ADOxx Meta2-Model can be found in [Bartho and Zivkovic, 2009].

Models are instances of metamodels which are instances of metametamodels. They are arranged in a hierarchy of 4 layers. Figure 2 depicts such a hierarchy. Here the Ecore metametamodel is chosen to define a metamodel for a process language or a metamodel for the *Physical Device DSL* (PDDSL), which is built by the language designer. He uses the metametamodel by creating instances of the concepts it provides. The language user takes into account the metamodel and creates instances which built a concrete process model or physical device model, respectively. Both models could represent systems running in the real world.

3 Combining Metamodels and Ontology Languages

Figure 3 depicts the general architecture of a bridge combining language software language metamodels and ontology technologies. The bridge itself is defined at the M3 layer, where a metametamodel like Ecore is considered and bridged with the OWL metamodel. Here we differentiate between two kinds of bridges: *M3 Integration Bridge* and *M3 Transformation Bridge*. In general, integration bridges are used to extend the expressiveness of a language, while the transformation bridge produces a new model based on a stable source language. In contrast to Section 4 where we present M2 bridges, M3 bridges are used to define syntactic expressions.

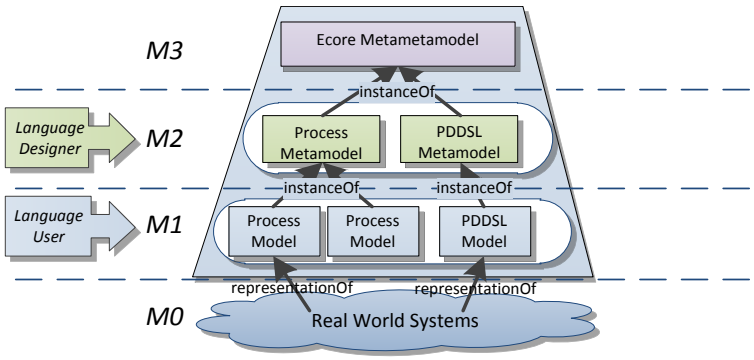

Figure 2: A metamodel hierarchy

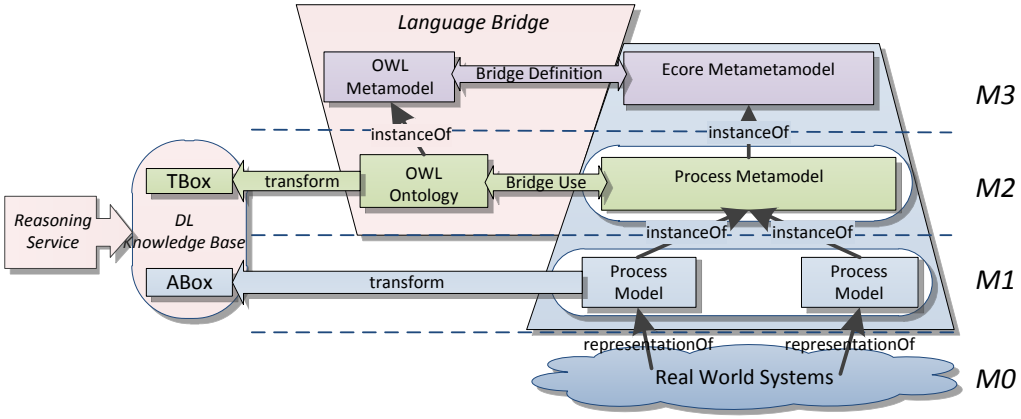

Figure 3: Language bridge

3.1 M3 Integration Bridge

The design of an M3 integration bridge consists mainly of identifying concepts in the Ecore metametamodel and the OWL metamodel, which are combined.

Here existing metamodel integration approaches (e.g. presented in [Walter and Ebert, 2009] and [Parreiras and Walter, 2008]) to combine the different metamodels are used. Result is a new metamodeling language, which allows for designing language metamodels at the M2 layer with integrated constraints. Such design and the benefits of integrated metamodels are exemplified in Section 3.1.1.

An integrated metamodeling language provides all classes of the Ecore metametamodel and OWL metamodel. It merges for example, OWL Class with Ecore EClass, OWL ObjectProperty with Ecore References or OWL DataProperty with Ecore Attribute. Thus, a strong connection between the two languages is built. Since a language designer creates a class, he is in the scope of both, OWL class and ECore class. Hence a language designer can use the designed class within OWL class axioms and simultaneously use features of the Ecore metamodeling language, like the definition of simple references between two classes.

The integration bridge itself is used at the M2 layer by a language designer. He is now able to define language metamodels with integrated OWL annotations to restrict the use of concepts he modeled and to extend the expressiveness of the language.

To provide modeling services to language user and language designer, the integrated metamodel is transformed into a Description Logics TBox. The models created by the language users are transformed into a corresponding Description Logics ABox. Based on the knowledge base consisting of TBox and ABox we can provide standard reasoning services and application specific modeling services to both language user and designer.

3.1.1 M3 Integration Bridge Physical Device Modelling

Having an integrated metametamodel available a language designer now can create language metamodels with integrated OWL constraints and axioms like the one depicted in Figure 4.

Overall the language designer has an ontology-based metamodeling language which provides a seamless and integrated design of formal syntactic constraints within the metamodel itself using some natural to use and simple to learn ontology languages, which can be used in combination with other, more familiar concrete syntaxes (e.g. with textual Ecore modeling syntax).

The metamodel together with instances (concrete PDDSL models) are transformed to an ontology which acts as input for reasoners. Here different services can provided to answer the key questions mentioned in Section 2.1.1. A full list of the services is given in [Wende, 2009].

3.2 M3 Transformation Bridge

The M3 Transformation Bridge allows language designers and language users to achieve representations of software languages (Metamodel/Model) in OWL. It provides the transformation of software language constructs like classes and properties into corresponding OWL constructs.

As one might notice, Ecore and OWL have a lot of similar constructs like classes, attributes and references. To extend the expressiveness of Ecore with OWL constructs, we need to establish mappings between the Ecore constructs onto OWL constructs. Table 3.2 presents a complete list of similar constructs.

```
class Cisco7603 equivalentWith restrictionOn hasConfiguration
with min 1 Configuration7603 {
}
class Configuration extends IntersectionOf(restrictionOn hasSlot with min 1
Slot, restrictionOn hasSlot with some restrictionOn hasCard with some
SuperVisor720) {
  reference hasSlot : Slot;
}
class Configuration7603 extends Configuration, equivalentWith
IntersectionOf(restrictionOn hasSlot with exactly 3 Slot, restrictionOn hasSlot
with some restrictionOn hasCard with some UnionOf(HotSwappableOSM,
SPAinterfaceProcessors) {
}
class Slot {
  reference hasCard [1-*]: Card;
}
class Card {
}
class SuperVisor720 extends Card {
}
class SPAinterfaceProcessors extends Card {
}
class HotSwappableOSM extends Card {
}
```

Figure 4: PDDSL metamodel with integrated ontologies at M2 layer

Table 1: Ecore and OWL: comparable constructs

Ecore	**OWL**
package	ontology
class	class
instance and literals	individual and literals
reference, attribute	object property, data property
data types	data types
enumeration	enumeration
multiplicity	cardinality

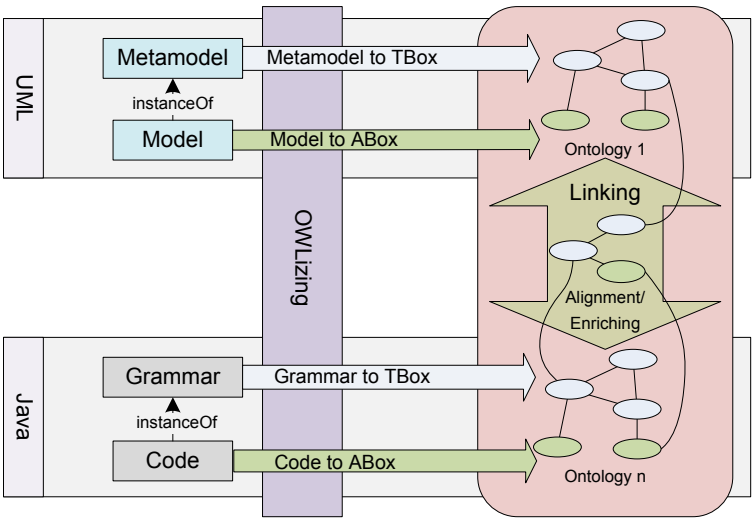

Figure 5: Aligning UML diagrams and Java code with OWL

Based on this mapping, we develop a generic transformation script to transform any Ecore Metamodel/Model into OWL TBox/ABox – OWLizer.

A model transformation takes for example the UML metamodel [OMG, 2007b] as input and generates an OWL ontology where the concepts, enumerations, properties and data types (TBox) correspond to classes, enumerations, attributes/references and data types in the UML metamodel. Another transformation takes the models (e.g. class diagrams, etc.) conforming to the UML metamodel created by the UML user and generates individuals in the same OWL ontology.

As one may notice, this is a generic approach to be used with any Ecore-based language. For example, one might want to transform the UML Metamodel/Models as well as all the Java grammar/code into OWL (classes/individuals). This approach can be seen as a linked data driven software development environment [Iqbal et al., 2009] and it is illustrated at Section 3.2.1.

3.2.1 M3 Transformation Bridge Application

Software development consists of multiple phases, from inception to production. During each software development phase, developers and other actors generate many artifacts, e.g. documents, models, diagrams, code, tests and bug reports. Although some of these artifacts are integrated, they are usually handled as islands inside the software development process.

Many of these artifacts (graphical or textual) are written using a structured language, which has a defined grammar. In a model-driven environment, concepts of software languages are represented by metamodels, whereas the artifacts written in those software languages are represented by models, which are described by the language metamodel. Thus, by transforming software metamodels and models into OWL and by aligning the OWL ontologies corresponding to software languages, we are able to link multiple data sources of a software development process, creating a linked-data repository for software development.

Let us consider an example of integrating two data sources: UML diagrams and Java Code.

Regardless of generating Java code from UML diagrams, developers would like to have a consistent view of corresponding classes and methods in UML and Java, i.e., developers might want to consult UML diagrams looking for a corresponding Java class. In this scenario, OWL and ontology technologies play an important role.

Figure 5 depicts the usage of M3 transformations together with ontology technologies. UML metamodel and model as well as Java grammar (metamodel) and java code (model) are transformed into OWL ontologies. Ontology alignment techniques [Euzenat and Shvaiko, 2007] can be used to identify some concepts in common between the two ontologies (UML and Java), e.g., package, class, method. Moreover, individuals with the same name in these two ontologies are likely the same.

Once the two ontologies are aligned, queries against the Java ontology also retrieve elements defined in UML diagrams. Now it is possible to retrieve sequence diagrams including a given Java class, since the two artifacts (UML diagrams and Java code) are now linked. This is only one example of the great potential provided by linking software engineering artifacts using OWL technologies.

4 Combining Models and Ontology Languages

Model bridges connect software models and ontologies on the modeling layer M1. They are defined in the metamodeling layer M2 between different metamodels. Figure 6 visualises a model bridge. The bridge is defined between a process metamodel on the software modeling side and an OWL metamodel in the OWL modeling hierarchy. The process metamodel is an instance of an Ecore (EMOF) metametamodel. In contrast to Section 4 where we presented M3 bridges which are used to define syntactic expressions, here in this section M2 bridges are used to reason (partially) on the formal semantics a modeling language has.

A model bridge is defined as follows: (1) Constructs in the software modeling and in the ontology space are identified. These constructs or language constructs used to define the corresponding models in the modeling layer M1. (2) Based on the identification of the constructs, the relationship between the constructs is analyzed and specified, i.e. the relationship of an Activity in a process metamodel like the BPMN metamodel to an OWL class. We distinguish between a *transformation* and *integration bridge*.

4.1 M2 Integration Bridge

Integration bridges merge information of the models from the software modeling and from the ontology space. This allows the building of integrated models (on modeling layer M1) using constructs of both modeling languages in a combined way, e.g. to integrate UML class diagrams and OWL.

As mentioned in section 3.2, UML class-based modeling and OWL comprise some constituents that are similar in many respects like classes, associations, properties, packages, types, generalization and instances [OMG, 2008]. Since both approaches provide complementary benefits, contemporary software development should make use of both. The benefits of an integration are twofold. Firstly, it provides software developers with more modeling power. Secondly, it enables semantic software developers to use object-oriented concepts like inheritance, operation and polymorphism together with ontologies in a platform independent way.

Such an integration is not only intriguing because of the heterogeneity of the two modeling approaches, but it is now a strict requirement to allow for the development of software with many

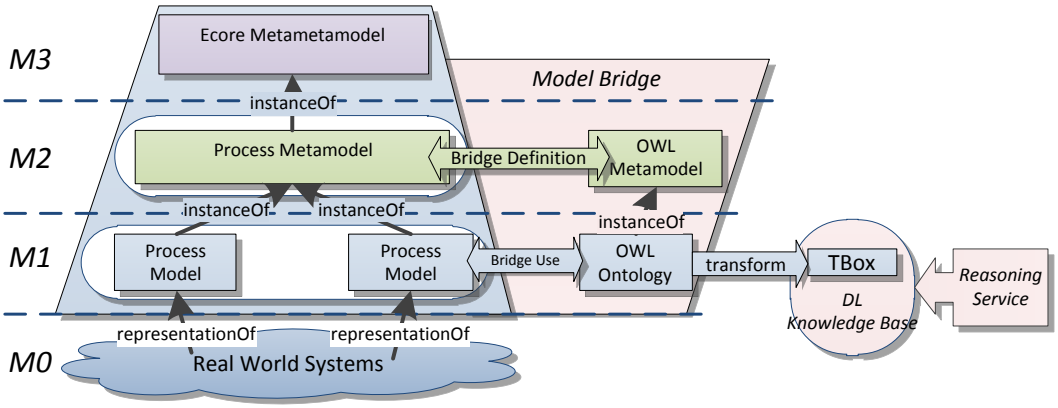

Figure 6: Model bridge

thousands of ontology classes and multiple dozens of complex software modules in the realms of medical informatics [O'Connor et al., 2007], multimedia [Staab et al., 2008] or engineering applications [Staab et al., 2006].

TwoUse (Transforming and Weaving Ontologies and UML in Software Engineering) addresses these types of systems [Silva Parreiras and Staab, 2010]. It is an approach combining UML class-based models with OWL ontologies to leverage the unique and potentially complementary strengths of the two. TwoUse consists of an integration of the MOF-based metamodels for UML and OWL, the specification of dynamic behavior referring to OWL reasoning and the definition of a joint profile for denoting hybrid models as well as other concrete syntaxes.

Figure 7 presents a model-driven view of the TwoUse approach. TwoUse uses UML profiled class diagrams as concrete syntax for designing combined models. The UML class diagrams profiled for TwoUse are input for model transformations that generate TwoUse models conforming to the TwoUse metamodel. The TwoUse metamodel provides the abstract syntax for the TwoUse approach, since we have explored different concrete syntaxes. Further model transformations take TwoUse models and generate the OWL ontology and Java code.

TwoUse allows developers to raise the level of abstraction of business rules previously embedded in code. It enables UML modeling with semantic expressiveness of OWL DL. TwoUse achieves improvements on the maintainability, reusability and extensibility for ontology based system development.

4.1.1 M2 Integration Bridge Application

In general, the Strategy Pattern solves the problem of dealing with variations. However, as already documented by [Gamma et al., 1995], 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.

The basic idea lies in decoupling class selection from the definition of client classes by exploiting OWL-DL modeling and reasoning. We explore a slight modification of the Strategy Pattern that includes OWL-DL modeling and that leads us to a minor, but powerful variation of existing practices: the Selector Pattern.

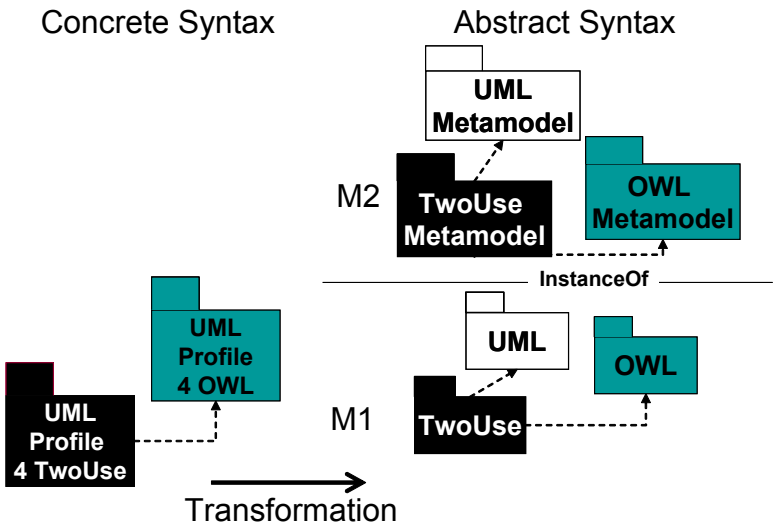

Figure 7: Model bridge

To integrate the UML class diagram with patterns and the OWL profiled class diagram , we rely on the TwoUse approach. The hybrid diagram is depicted in Figure 8. The Selector Pattern is composed by a *context*, the specific *variants* of this context and their respective descriptions, and the *concept*, which provides a common interface for the variations (Figure 8). Its participants are:

- `Context` maintains a reference to the `Concept` object.
- `Concept` declares an abstract method `behavior` common to all variants.
- `Variants` implement the method `behavior` of the class `Concept`.

The *Context* has the operation `select`, which uses OWL-like query operations to dynamically classify the object according to the logical descriptions of the variants. A *Variant* is returned as result (Figure 8). Then, the *Context* establishes an association with the *Concept*, which interfaces the variation.

The application of the Selector Pattern presents some consequences, that we discuss as follows:

Reuse. The knowledge represented in OWL-DL can be reused independently of platform or programming language.

Flexibility. The knowledge encoded in OWL-DL can be modeled and evolved independently of the execution logic.

Testability. The OWL-DL part of the model can be automatically tested by logical unit tests, independently of the UML development.

The application of TwoUse can be extended to other design patterns concerning variant management and control of execution and method selection. Design patterns that factor out

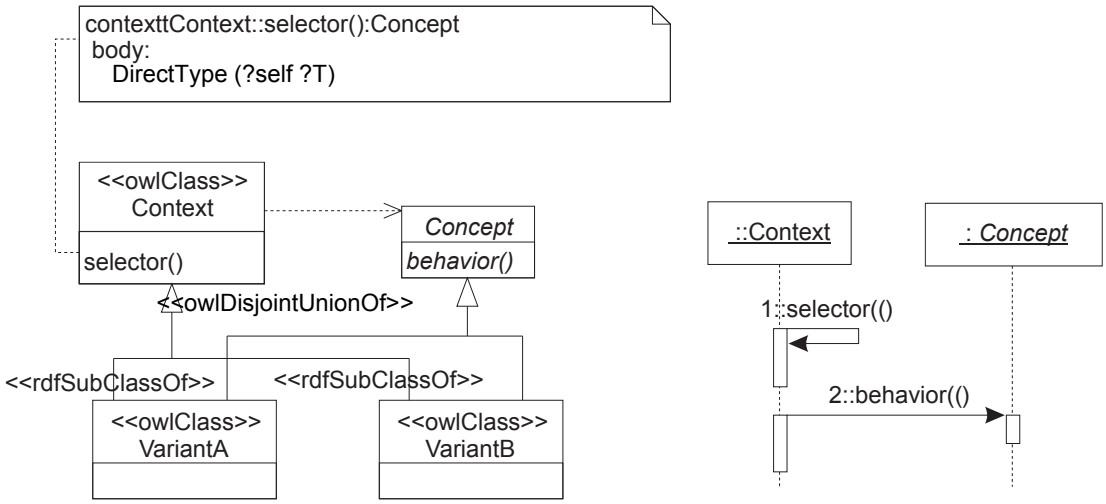

Figure 8: The Selector Pattern

commonality of related objects, like Prototype, Factory Method and Template Method, are good candidates.

4.2 M2 Transformation Bridge

A *transformation bridge* describes a (physical) transformation between models in layer M1. The models are kept separately in both modeling spaces. The information is moved from one model to the model in the other modeling space according to the transformation bridge. With respect to the example depicted in Figure 6, a process model like a UML Activity Diagram is transformed to an OWL ontology. The transformation rules or patterns are defined by the bridge.

4.2.1 M2 Transformation Bridge Application

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. Both metamodels are instances of Ecore metametamodels. 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.

We identified the following shortcomings of process models in the software modeling space. (1) A semantic representation of control flow dependencies of activities in a process, i.e. execution ordering of activities in a flow. Such constraints allow the description of order dependencies like an activity requires a certain activity as a successor. (2) It is quite common in model-driven engineering to specialize or refine a model into a more fine-grained representation that is closer to the concrete implementation. In process modeling, activities could be replaced by subactivities for a more precise description of a process. Hence, modeling possibilities for subactivities and also for control flow constraints of these subactivities are a relevant issue. (3) Quite often,

Construct	UML Notation	DL Notation
1. Start	●	$Start_i$
2. End	◉	End_i
3. Activity	Receive Order	*Receive Order*
4. Edge	⟶	TO_i
5. Process P	Receive Order	$P \equiv Start_i \sqcap \exists_{=1} TO_i.$ $(ReceiveOrder \sqcap \exists_{=1} TO_i.End_i)$
6. Flow	Receive Order → Fill Order	$ReceiveOrder \sqcap \exists_{=1} TO_i.FillOrder$
7. Decision	Receive Order; Reject Order / Fill Order; Close Order	$ReceiveOrder \sqcap \exists_{=1} TO_i.$ $((RejectOrder \sqcup FillOrder)$ $\sqcap \exists_{=1} TO_i.CloseOrder)$
8. Condition	Receive Order → [Order accepted] → Fill Order	$ReceiveOrder \sqcap \exists_{=1} TO_i.$ $((FillOrder \sqcap \kappa_{orderaccepted}) \sqcup$ $(Stalled \sqcap \neg\kappa_{orderaccepted}))$
9. Fork and Join	Receive Order; Ship Order / Send Invoice; Close Order	$ReceiveOrder \sqcap \exists TO_i.$ $(ShipOrder \sqcap \exists_{=1} TO_i.CloseOrder)$ $\sqcap \exists TO_i.(SendInvoice \sqcap$ $\exists_{=1} TO_i.CloseOrder) \sqcap = 2\ TO_i$
10. Loop	Receive Order → Fill Order	$Loop_j \sqcap \exists_{=1} TO_i.FillOrder,$ $Loop_j \equiv ReceiveOrder \sqcap \exists_{=1} TO_j.$ $(Loop_j \sqcup End_j)$

Table 2: Transformation to OWL.

one may formulate process properties that cover modality, i.e. to express a certain property like the occurrence of an activity within a control flow is optional or unavoidable in all possible process instances (traces).

In the following we give an overview of the model bridge from a UML activity diagram to an OWL ontology with a short discussion of design decisions. In addition, we demonstrate the usage of ontological representation of a process model in order to compensate the aforementioned shortcomings in software process modeling.

Process Modeling Principles in OWL A process model describes the set of all process runs or traces it allows. Activities are represented by OWL classes and a process is modeled as a complex expression that captures all activities of the process. A process run is an instance of this complex class expression in OWL. The process models are described in OWL DL, as syntax we use the DL notation. Transformation patterns from UML activity diagrams to OWL are given in Tab. 2.

Control flow relations between activities are represented by object properties in OWL, i.e. by the property TO_i. All TO_i object properties are subproperties of the transitive property TOT. A process is composed by activities which is described in OWL by axioms as show in No. 5. The control flow (No. 6) is a class expression in OWL like $A \sqcap \exists TO_i.B$ that means

the activity A is directly followed by the activity B. We use concept union for decisions (No. 7). The non-deterministic choice between activity B and C is given by the class expression $\exists TO_i.(B \sqcup C)$. Flow conditions (No. 8) are assigned to the control flow. A loop (No.10) is a special kind of decision. An additional OWL class $Loop_j$ for the subprocess with the loop is introduced to describe multiple occurrences of the activities within the loop. Parallel executions are represented by intersections (No.9). It is an explicit statement that an activity have multiple successors simultaneously.

Process Modeling and Retrieval in OWL The semantic representation of process models in OWL tackles the problems and shortcoming that are mentioned at the beginning of this section in multiple ways. For instance the validation of process properties, specializations or refinement relations between process models, as well as the retrieval of processes benefits from this representation. This section gives an overview of query patterns for process retrieval with semantic queries.

The process model in OWL gives an explicit description of the execution order dependencies of activities. 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.

5 Achievements: The TwoUse Toolkit

We have realized the approaches described in WP1 deliverables in the TwoUse Toolkit[1] It is an open source tool that implements current OMG and W3C standards for developing ontology-based software models and model-based OWL ontologies. It is a model-driven tool to bridge the gap between Semantic Web and Model Driven Software Development.

TwoUse toolkit building blocks are:

- A set of model transformations. Generic transformations like Ecore2OWL and XML2OWL (Language Bridges) allows developers to transform any software language into OWL. Specific transformations like UML2OWL and BPMN2OWL (Model Bridges) allow developers to OWL representations of software models.

- A set of textual and graphical editors. TwoUse relies on textual and graphical editors for editing and parsing W3C standard languages like OWL2 and SPARQL, OMG stan-

[1] `http://twouse.googlecode.com/`.

dards like UML and OCL (Deliverable D1.1 [Parreiras and Walter, 2008]) as well as other domain-specific languages (Deliverable D1.2 [Zhao et al., 2009]).

- A set of reasoning services like classification, realization, query answering and explanation (Deliverable 2.5.1 [Wende, 2009]).

The TwoUse Toolkit has two user profiles: model-driven software developer and OWL ontology engineer. TwoUse enables model-driven software developers with the following functionalities:

- Describe classes in UML class diagrams using OWL class descriptions (D1.1 [Parreiras and Walter, 2008]).
- Semantically search for classes, properties and instances in UML class diagrams (D1.2 [Zhao et al., 2009]).
- Design business rules using the UML Profile for SWRL (D1.1 [Parreiras and Walter, 2008]).
- Extent software design patterns with OWL class descriptions (D1.1 [Parreiras and Walter, 2008]).
- Make sense of UML class diagrams using inference explanations (D2.5.1 [Wende, 2009]).
- Link software engineering artifacts by transforming software languages into OWL (OWL-izing) (Language Bridge).
- Write OWL queries using SPARQL or the OWL-like languages using query editors with syntax highlighting (D1.2 [Zhao et al., 2009]).
- Validate refinements on business process models (D3.4 [Zhao et al., 2010]).

OWL ontology engineers are able to:

- Graphically model OWL ontologies and OWL safe rules using OMG UML Profile for OWL and UML Profile for SWRL [OMG, 2008].
- Graphically model OWL ontologies and OWL Safe Rules using the OWL Graphical Editor.
- Graphically model and store ontology design patterns as templates [Silva Parreiras et al., 2010].
- Write and safe SPARQL and SPARQLAS queries using the textual editors with syntax highlighting [Schneider, 2010].

6 Conclusion

In this deliverable, we presented the building blocks for bridging software languages and ontology technologies in MOST. Language bridges are generic and can be used in existing software languages as well as new software languages that explore the extended functionalities provided by OWL. Model bridges have an ad-hoc character and are language specific.

While language bridges improve software development by realizing some of the major motivations of the OWL language, e.g., shared terminology, evolution, interoperability and inconsistency detection, model bridges allow for exploring new ways of modeling software as well as different ways of exploiting reasoning technologies.

In particular we presented different bridges. Here, *integration bridges* are used to extend the expressiveness of a language, while the *transformation bridge* produces a new model based on a stable source language. *M3 bridges* are used to define syntactic expressions, while *M2 bridges* are used to reason (partially) on the formal semantics. These bridges are used in ontology driven software development for combining modeling and metamodeling languages with ontology technologies to reason on syntax and semantics of modeling languages.

References

[Atkinson and Kühne, 2003] Atkinson, C. and Kühne, T. (2003). Model-driven development: a metamodeling foundation. *IEEE software*, 20(5):36–41.

[Bartho and Zivkovic, 2009] Bartho, A. and Zivkovic, S. (2009). Modeled software guidance/engineering processes and systems. Deliverable ICT216691/TUD/WP2-D2/D/PU/b1.00, Technial University Dresden, BOC. EU FP7 STREP MOST Project number ICT-2008-216691.

[Bildhauer et al., 2008] Bildhauer, D., Riediger, V., Schwarz, H., and Strauss, S. (2008). grUML-An UMLbased Modeling Language for TGraphs. to appear in Arbeitsberichte Informatik, Universität Koblenz-Landau.

[Budinsky et al., 2003] Budinsky, F., Brodsky, S., and Merks, E. (2003). *Eclipse modeling framework*. Pearson Education.

[Ebert, 2008] Ebert, J. (2008). Metamodels Taken Seriously: The TGraph Approach. In Kontogiannis, K., Tjortjis, C., and Winter, A., editors, *12th European Conference on Software Maintenance and Reengineering*, Piscataway, NJ. IEEE Computer Society.

[Euzenat and Shvaiko, 2007] Euzenat, J. and Shvaiko, P. (2007). *Ontology matching*. Springer-Verlag, Heidelberg.

[Gamma et al., 1995] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional.

[Iqbal et al., 2009] Iqbal, A., Ureche, O., Hausenblas, M., and Tummarello, G. (2009). Ld2sd: Linked data driven software development. In *Proceedings of the 21st International Conference on Software Engineering & Knowledge Engineering (SEKE'2009), Boston, Massachusetts, USA, July 1-3, 2009*, pages 240–245. Knowledge Systems Institute Graduate School.

[Mellor et al., 2003] Mellor, S., Clark, A., and Futagami, T. (2003). Model-driven development. *IEEE software*, 20(5):14–18.

[Motik et al., 2009] Motik, B., Patel-Schneider, P. F., and Horrocks, I. (2009). OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax. `http://www.w3.org/TR/owl2-syntax/`.

[O'Connor et al., 2007] O'Connor, M. J., Shankar, R., Tu, S. W., Nyulas, C., Parrish, D., Musen, M. A., and Das, A. K. (2007). Using semantic web technologies for knowledge-driven querying of biomedical data. In *AIME*, pages 267–276.

[OMG, 2006] OMG (2006). Meta Object Facility (MOF) Core Specification. `http://www.omg.org/docs/formal/06-01-01.pdf`.

[OMG, 2007a] OMG (2007a). UML Infrastructure Specification, v2.1.2. *OMG Adopted Specification*.

[OMG, 2007b] OMG (2007b). *Unified Modeling Language: Superstructure, version 2.1.1*. Object Modeling Group.

[OMG, 2008] OMG (2008). *Ontology Definition Metamodel*. Object Modeling Group.

[Parreiras and Walter, 2008] Parreiras, F. S. and Walter, T. (2008). Report on the combined metamodel. Deliverable ICT216691/UoKL/WP1-D1.1/D/PU/a1, University of Koblenz-Landau. MOST Project.

[Parreiras et al., 2009] Parreiras, F. S., Walter, T., and Wende, C. (2009). Report on transformation patterns. Deliverable ICT216691/UoKL/WP1-D1.3/D/PU/a1, University of Koblenz-Landau. MOST Project.

[Schneider, 2010] Schneider, M. (2010). SPARQLAS - Implementing SPARQL Queries with OWL Syntax. In *Proceedings of the Third Workshop on Transforming and Weaving Ontologies and Model Driven Engineering (TWOMDE 2010), 30 June, Malaga, Spain*, number 604. CEUR-WS.org.

[Silva Parreiras et al., 2010] Silva Parreiras, F., Groener, G., Walter, T., and Staab, S. (2010). A Model-Driven Approach for Using Templates in OWL Ontologies. In *Knowledge Management and Engineering by the Masses, 17th International Conference, EKAW 2010, Lisbon, Portugal, October 11 - 15, 2010. Proceedings*, volume 6317 of *LNAI*, pages 350–359. Springer.

[Silva Parreiras and Staab, 2010] Silva Parreiras, F. and Staab, S. (2010). Using ontologies with uml class-based modeling: The twouse approach. *Data Knowl. Eng.* To be published.

[Staab et al., 2006] Staab, S., Franz, T., Görlitz, O., Saathoff, C., Schenk, S., and Sizov, S. (2006). Lifecycle knowledge management: Getting the semantics across in x-media. In *Foundations of Intelligent Systems, ISMIS 2006, Bari, Italy, September 2006*, volume 4203 of *LNCS*, pages 1–10. Springer.

[Staab et al., 2008] Staab, S., Scherp, A., Arndt, R., Troncy, R., Gregorzek, M., Saathoff, C., Schenk, S., and Hardman, L. (2008). Semantic multimedia. In *Reasoning Web, 4th International Summer School, Venice, Italy*, volume 5224 of *LNCS*, pages 125–170. Springer.

[Walter and Ebert, 2009] Walter, T. and Ebert, J. (2009). Combining DSLs and Ontologies using Metamodel Integration. In *Domain-Specific Languages*, volume LNCS, pages 148–169. Springer.

[Wende, 2009] Wende, C. (2009). Ontology Services for Model-Driven Software Development. MOST Project Deliverable. `www.most-project.eu`.

[Zhao et al., 2010] Zhao, Y., , Wende, C., Pan, J. Z., Thomas, E., Gröner, G., Jekjantuk, N., Ren, Y., and Walter, T. (2010). Guidance tools for language transformations. Deliverable ICT216691/UNIABDN/WP3-D4/D/PU/b1, University of Aberdeen. MOST Project.

[Zhao et al., 2009] Zhao, Y., Pan, J. Z., Nophadol Jekjantuk, F. S. P., Gröner, G., and Walter, T. (2009). Report on querying the combined metamodel. Deliverable ICT216691/UoKL/WP1-D1.2/D/PU/a1, University of Koblenz-Landau. MOST Project.