Automatic Generation and Evolution of Model Transformations Using Ontology Engineering Space

Stephan Roser and Bernhard Bauer

Programming of Distributed Systems
Institute of Computer Science, University of Augsburg, D-86135 Augsburg, Germany
{roser,bauer}@ds-lab.org

Abstract. Model-driven software development facilitates faster and more flexible integration of information and communication systems. It divides system descriptions into models of different view points and abstraction levels. To effectively realize cross-organisational collaborations, it is an important prerequisite to exchange models between different modelling languages and tools. Knowledge is captured in model transformations, which are continuously adjusted to new modelling formats and tools. However, interoperability problems in modelling can hardly be overcome by solutions that essentially operate at syntactical level. This paper presents a novel approach using the capabilities of semantic technologies in order to improve cross-organisational modelling by automatic generation and evolution of model transformations.

1 Introduction

New challenges arise with the development of increasingly complex systems across enterprises. Systems must be more adaptable to changing requirements. It has to be possible to compose systems from existing components, to replace parts of systems, and to integrate existing systems. Model-driven software development (MDSD) is a young but promising approach to deal with these challenges. However, to enable an efficient development of flexible cross-organisational information and communication systems one needs to support interoperability in modelling enterprises and applications. As enterprises often apply different methodologies, they need to share their enterprise models and knowledge independent of languages and tools. Therefore, one needs to develop mappings between different existing enterprise modelling formalisms based on an enterprise modelling ontology as well as tools and services for translating models (IDEAS analysis - gap 12 [21]). The IDEAS network stated in its vision for 2010 [22] requirements to enable enterprises to seamlessly collaborate with others. According to this, it is necessary to integrate and adapt ontologies in architectures and infrastructures to the layers of enterprise architecture and to operational models. This can be done by applying mappings between different enterprise model formalisms based on an enterprise modelling ontology. Heterogeneous business models can be semantically enriched by ontologies to achieve a shared understanding of the enterprise domain.

[10] proposes a rather abstract interoperability framework for MDSD of software systems, which supports the business interoperability needs of an enterprise. Mutual understanding on all levels of integration, conceptual, technical, and applicative level, has to be achieved. One uses the conceptual reference model to model interoperability, whereas metamodels and ontologies are used to define model transformations and model mappings between the different views of an enterprise system. Still, solutions like [5], [27], and [41], that aim at improving such kind of interoperability, address the problems of different representation formats, modelling guidelines, modelling styles, modelling languages, and methodologies at the syntactical level. Their focus is on metamodels' abstract and concrete syntax. Approaches providing interoperability solutions based on ontologies and automated reasoning lack key features for modelling [16]. For example they do not store trace information of transformation executions in order to enable transactions or incremental updates [37].

In this work we propose the approach of *ontology-based model transformation* (ontMT). It integrates ontologies in modelling by utilising different technological spaces [26] (namely MDA and Ontology technological space) to automate the generation and evolution of model transformations. Interoperability in modelling is fostered by employing automated reasoning technologies from the ontology engineering technological space for the generation of model transformations. We present how the ontMT approach can be realized as a *semantic-enabled model transformation tool (Sem-MT-Tool)* in a semantic-enabled modelling and development suite (see [2]). This tool applies technology bridging MDA and Semantic Web approaches and makes use of the capabilities and benefits of both approaches.

This paper is organized as follows: After introducing background information in Section 2, we provide a problem description in Section 3. The approach of ontology-based model transformation is presented in Section 4. Section 5 provides insights into the components of a semantic-enabled modelling and development tool realizing the ontMT approach. Section 6 contains a detailed case study. Section 7 discusses the ontMT approach and Section 8 provides related work. Finally, we conclude with a short summary and outlook in Section 9.

2 Background and Context

Model-driven Software Development (MDSD): MDSD, as a generalization of OMG™'s Model Driven Architecture paradigm (MDA®), is an approach to software development based on modelling and automated transformation of models to implementations [15]. In MDSD models are more than abstract descriptions of systems: they are the key part in defining software systems, as they are used for model- and code generation. Largely automated model transformations refine abstract models to more concrete models or simply describe mappings between models of the same level of abstraction. As model transformations play a key role in MDSD, it is important that transformations can be developed as efficiently as possible [16].

Models: The definition of the mega-model[1] presented in [14] describes a model as a system that enables us to give answers about a system under study without the need to

[1] Models about modelling are called mega-models, while metamodels are models of modelling languages.

consider this system directly. In short, a model is a *representation of* a system, whereas systems can be physically observable elements or more abstract concepts like modelling languages. A modelling language is a set of models, and models are *elements of* a modelling language. Models *conform to* a model of the modelling language, i.e. a metamodel. Those metamodels can be used to validate models. For one modelling language multiple (meta)models can exist, which can differ in the language they are described in.

Model transformations: Model transformations (MTs) are specified between metamodels. The execution of a model transformation transforms models conforming to the source metamodel into models conforming to the target metamodel. *Vertical model transformations* refine abstract models to more concrete models, while *horizontal model transformations* describe *mappings* between models of the same abstraction level. With the MOF 2.0 Query/View/Transformation specification [37] (QVT) the OMG provides a standard syntax and execution semantics for transformations used in a MDSD tools chain. The QVT relations language allows to specify relationships between MOF models declaratively. It supports complex object pattern matching and implicitly traces the transformation execution. A relational transformation defines how a set of models can be transformed into another. Relations in a transformation declare constraints that must be satisfied by the elements of the candidate models. Domains are part of a relation and have patterns. Patterns can be considered as templates for objects and their properties that must be located, modified, or created in a candidate model that should satisfy the relation. A domain pattern consists of a graph of object template expressions (OTEs) and property template items (PTIs). An OTE specifies a pattern that matches model elements. It is uses a collection of PTIs to specify constraints on the values of the properties of model elements.

```
relation PackageToSchema {
  domain uml p:Package {name=pn}
  domain rdbms s:Schema {name=pn}
}
```

Listing 1: QVT sample relation

Listing 1 depicts a relation specified in QVT relational syntax [37, p.13]. Two domains are declared that match elements of the *uml* and *rdbms* models respectively. Each domain specifies a pattern: a *Package* with a name, and a *Schema* with a name. Both *name* properties being bound to the same variable *pn* implying that they should have the same value.

Ontology: Ontologies are considered a key element for semantic interoperability. They act as shared vocabularies for describing the relevant notions of application areas, whose semantics is specified in a (reasonably) unambiguous and machine-processable form [7]. According to [33] an ontology differs from existing methods and technologies in the following way: (i) the primary goal of ontologies is to enable agreement on the meaning of specific vocabulary terms and to facilitate information integration across individual languages. (ii) Ontologies are formalized in logic-based representation languages. Thus, their semantics is specified in an unambiguous way. (iii) The representation languages come with executable calculi enabling querying and reasoning at run time. *Application ontologies* contain the definitions specific to a

particular application [19]. *Reference ontologies* refer to ontological theories, whose focus is to clarify the intended meanings of terms used in specific domains.

Technological Spaces: Kurtev et al. [26] introduce the concept of *technological spaces* (TS) aiming to improve efficiency of work by using the best possibilities of different technologies. A technological space is, in short, a zone of established expertise and ongoing research. It is a working context together with a set of associated concepts, body of knowledge, tools, required skills, and possibilities. Initially five technological spaces (MDA TS, XML TS, Abstract Syntax TS, Ontology TS, DBMS TS) have been presented in [26], of which the *MDA TS* and the *Ontology TS* are important for our work. In the MDA TS models are considered as first-class citizens, that represent particular views on the system being built. The Ontology TS can be considered as a subfield of knowledge engineering, mainly dealing with representation and reasoning. The ontology engineering space performs outstanding in traceability, i.e. in the specification of correspondences between various metamodels, while the MDA TS is much more applicable to facilitate aspect or content separation. With the Ontology Definition Metamodel (ODM) [35] the OMG issues a specification defining a family of independent metamodels and mappings among these metamodels. These metamodels correspond to several international standards for ontology definition. ODM comprises metamodels for RDF(S), OWL, common logic (CL), topic maps (TM), and as a non normative part description logic (DL). Metamodels for RDF(S) and OWL represent more structural or descriptive representations, which are commonly used in the semantic web community. ODM further defines transformations between the UML2 metamodel and the OWL metamodel defined in ODM.

Semantics: The notion of the term semantics depends on the context it is used in and varies by the people using it. As the root of the problem Harel and Rumpe [20] identify insufficient regard for the crucial distinction between syntax and true semantics. Thus we clarify a few terms that have particular significance to this work.

- **Syntax:** *Syntax* N_L is the notation of a language *L*. It is distinguished between the concrete syntax, the textual or graphical representation of the language, and an abstract syntax or metamodel, being the machine's internal representation. A metamodel is a way to describe the syntax of a language [20].
- **Semantics:** Semantics is the meaning of language, that is expressed by relating the syntax to a semantic domain. The description of a *semantic domain S* (its notation is N_S) can vary from plain English to mathematics. Semantics is defined by a *semantic mapping M:* $L \rightarrow S$ from the language's syntax to its semantic domain [20].
- **Ontological:** According to [35] 'an ontology defines the common terms and concepts (meaning) used to describe and represent and area of knowledge'. Talking about 'ontological' we mean technology of the Ontology TS. That is to say technology based on logic like RDF(S) or OWL, which is used by the semantic web community to describe e.g. vocabularies or ontologies.

3 Problem Description

To enable collaboration in enterprise and systems modelling, enterprises have to be supported by interoperability solutions for model sharing and model exchange independent of modelling languages and tools. Also the evolution of model transformations

has to be considered. To maintain and reuse existing model transformations, model transformations have to be adjusted to new modelling languages or styles. This section illustrates these challenges via a MDSD scenario. It further discusses the problems that possible automation solutions face.

3.1 A MDSD Scenario

Figure 1 illustrates the application of MDSD to cross-organisational business process development. The vertical dimension distinguishes the different layers of abstraction applied in MDSD, and the horizontal dimension represents the collaborative modelling between two enterprises A and B. Models of enterprises A and B have to be shared at different levels of abstraction in order to agree on and develop cross-organisational business processes.

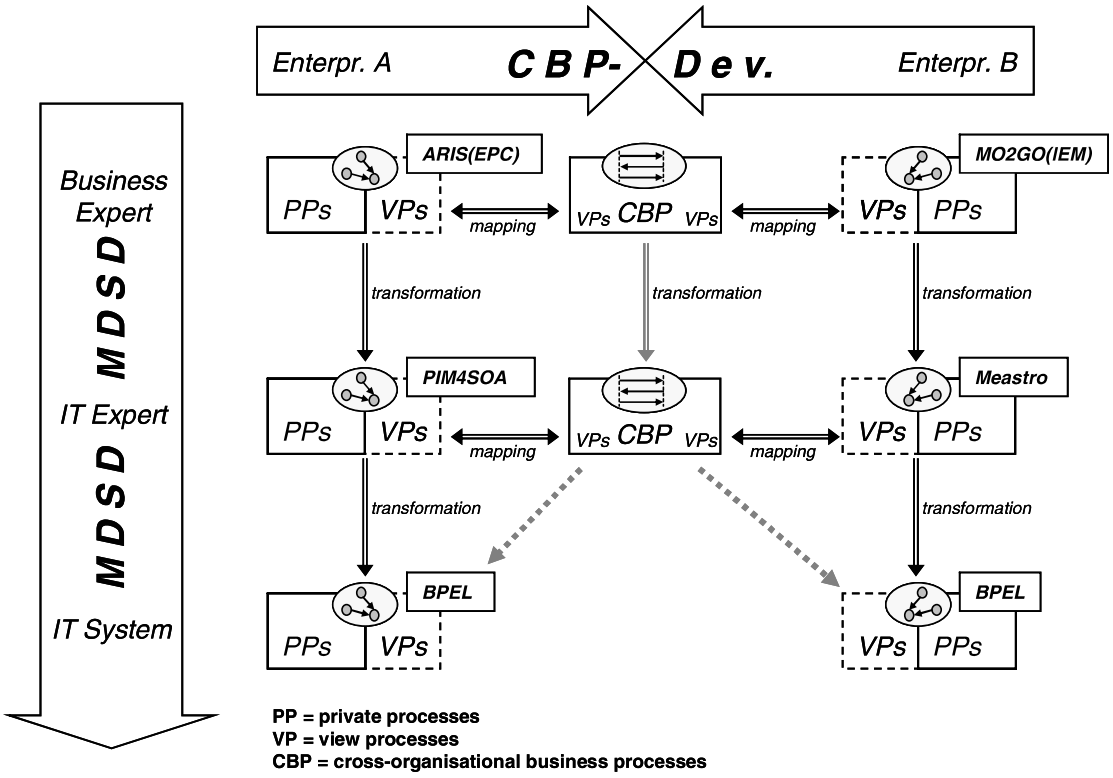

Fig. 1. Scenario realizing cross-organisational business process modelling and execution

A concrete scenario, implementing cross-organisational business process modelling and execution like show in Figure 1, has been developed in the ATHENA project (more details can be found in [18]). Enterprises A and B develop models for their processes (privates processes (PPs), view processes (VPs), and cross-organisational business processes (CBPs)) at three levels of abstraction, i.e. business expert, IT expert, and IT-system level. Vertical transformations, like presented in [1], encode knowledge about the architecture and the platform in order to transform models from higher to lower abstraction level. For example ARIS models (eEPCs [25]) are transformed to models conforming to PIM4SOA [3]. Enterprises A and B use different modelling tools and languages at the various abstraction levels. To develop

cross-organisational business processes, both enterprises have to provide public parts of their models as a basis for discussion during collaborative modelling.

However, some issues prevent a smooth realization of such a MDSD scenario:

- **Exchange of models:** Models are shared across inter-organisational relationships. Hence, mappings have to be developed between the various enterprises' modelling languages and tools. This is necessary to achieve a shared understanding of cross-organisational business processes and to enable collaborative MDSD.
- **Evolution of model transformations:** Over a period of time enterprises will apply new (versions) of modelling languages, metamodels, and modelling styles. Therefore, existing transformations have to be maintained, adjusted, or redeveloped.

3.2 Problem Statement

Managing and developing model transformations are error-prone and long lasting tasks. Since model transformations are a kind of metaprogramming, they require a deep knowledge of all the underlying modelling technology, which is, in most cases, quite hard to learn. Thus, it is beneficial to provide support with a solution that automates model transformation development and adjustment tasks. Despite the multiplicity of model transformations and application scenarios, the core principles of modelling (i.e. representing information about real world things in models) and problems of such automation solutions remain the same. The core barriers to model exchange and maintenance of model transformations are multiple representation formats and different modelling styles, serving the particular application.

- **Different representation format:** The trend towards the use of domain specific languages (DSLs) leads more and more people to create their own domain specific models (DSMs). This naturally results in a variety of different languages and metamodels. To exchange models that conform to these various metamodels (abstract syntax), model transformations have to be developed. Often there are multiple model transformations for the same modelling language. Also time and again new versions of metamodels, e.g. the metamodels for UML 1.x and UML 2.x, are released. Whenever new versions replace the old ones, new model transformations have to be developed and existing model transformations have to be adjusted. Though visual representations (concrete syntax) should be decoupled from internal representation (abstract syntax), different concrete syntax is often considered in model transformations to provide e.g. views on models.
- **Different semantics:** Since the semantics of modelling languages' concepts is rarely formally specified (in the UML specification this is plain English), different people and organisations can associate different semantics with the same concepts used in the metamodel. This is often done by applying special modelling styles and representation guidelines. Again, model transformations have to be specified enabling sensible exchange of models according to the respective interpretations.

4 Ontology Supported Model Transformations

OntMT facilitates methods to generate and adjust model transformations despite of structural and semantic differences of metamodels. Different representation formats

and different semantics (as described in Section 3.2) are overcome by applying semantic web technology of the Ontology TS. In ontMT metamodels are annotated through the elements of a reference ontology (RO) and reasoning is applied to the RO and the annotations. OntMT allows to generate and adjust common model transformations automatically in order to apply MDSD in the MDA TS.

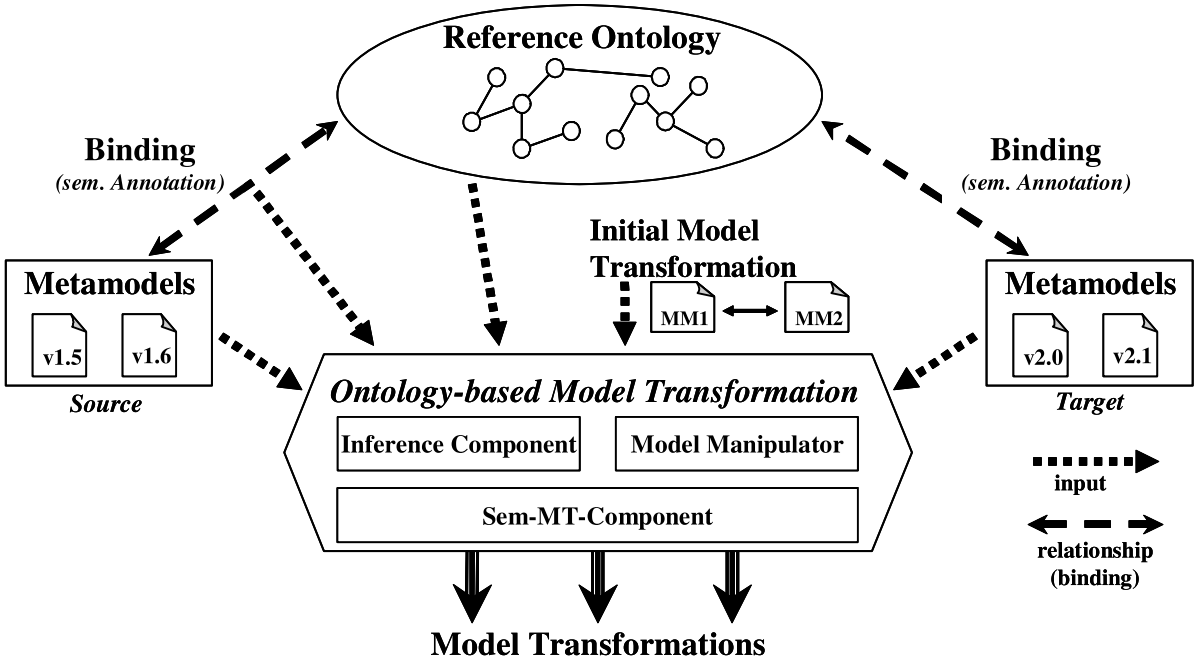

Fig. 2. Ontology-based model transformation – overall approach

Figure 2 depicts the overall approach of ontMT. Different versions of metamodels are bound to a reference ontology of a certain domain. Bindings (semantic annotations) specify the semantic mapping of metamodels to the semantics of their concepts, i.e. to the reference ontology. To generate model transformations for various model transformation languages and to adjust existing model transformations, ontMT makes use of reasoning mechanisms. The metamodels and the reference ontology are given, while the bindings of the metamodels to the reference ontology have to be specified. Finally, an initial model transformation is needed. For the evolution of model transformations the initial model transformation is the model transformation that shall be reused or adjusted (see Section 4.2). The initial model transformation (e.g. from metamodel v1.5 to metamodel v2.0) encodes transformation rules and especially the semantics of the model transformation. If for example the metamodel v2.0 is replaced with a version 2.1 only the delta between these metamodel has to be considered to adjust the existing model transformation. The new model transformation is generated by substituting the concepts of metamodel v2.0 with the concepts of metamodel v2.1 in the initial model transformation. In the case of automated mapping generation, a bootstrapping algorithm generates an initial model transformation (see Section 4.1).

4.1 Generation of Model Transformations

Model transformations between various modelling languages can be automatically derived and generated with the ontMT approach. In this section we describe the

procedure to generate mappings, i.e. semantically identical model transformations, between two modelling languages *A* and *B*. We illustrate the procedure via a strongly simplified example, where *A* and *B* both consist of two concepts: *A={Process, Task}* and *B={EPC, EPCElement}*.

For both languages exists an abstract syntax N_A/N_B in various technological spaces: A has (like B) an abstract syntax in the MDA TS $N_{A\text{-}mda}$ and the Ontology TS $N_{A\text{-}ont}$ which are synchronized. Thus, one can work with the syntax and the capability of that technological space, that is better suited for solving a problem (see Figure 3). The semantics of the concepts is described by the means of the semantic domain *SD* and its notation in a reference ontology N_{RO} (e.g. OWL) respectively. Semantics of the languages is defined by semantic mappings from the languages to the semantic domain: M_A*: A* $\rightarrow$ *SD* and M_B*: B* $\rightarrow$ *SD*. In this example, the semantic domain is given as *SD={Activity, Action}*, while the semantic mappings are M_A*={Process* $\cong$ *Activity, Task* $\cong$ *Action}* and M_B*={EPC* $\cong$ *Activity, EPCElement* $\cong$ *Action}*[2].

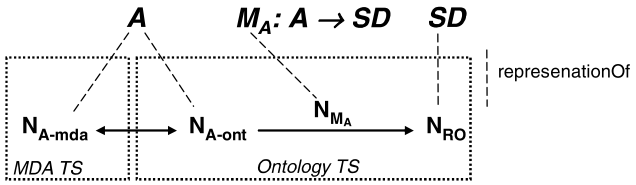

Fig. 3. Modelling language, semantic mapping, semantic domain and their representations

The ontological grounding[3] is a notation of the semantic mapping from $N_{A\text{-}ont}$ to N_{RO}. The goal of the transformation to generate is to define 'identity' relationships between the concepts of *A* and *B*. The model transformation MT_{mapAB}: *A* $\leftrightarrow$ *B* between *A* and *B* has the following semantics: $M_{MTmapAB}$*:* MT_{mapAB} $\rightarrow$ *id*, where *id* is the identical mapping. The generation procedure works on the model of the model transformation and the models of the modelling languages. It exploits the ontological grounding to the reference ontology. On the basis of reasoning results gained in the Ontology TS (*{Process* $\cong$ *EPC, Task* $\cong$ *EPCElement}*), modification operations are called to obtain the new model transformation working solely on the model of the model transformation. To generate the model transformation MT_{mapAB}, the following steps are performed (see Figure 4a):

- © A bootstrapping algorithm generates the model transformation MT_{mapAA}*: A* $\leftrightarrow$ *A*, which is a mapping of *A* on itself. This bootstrapping step is necessary to obtain a first model of the model transformation (transforming N_A to N_A')[4], which only has to be adjusted by modifications operations. Assuming the same ontological grounding for N_A and N_A', the bootstrapping model transformation is an *id*: M_{MTma}

[2] $\cong$ stands for equivalence.

[3] The definition of the ontological grounding is a semantic annotation comprising static semantics of the metamodels, i.e. the semantics of the concepts and an ontology respectively.

[4] Such a mapping can be generated on the basis of a metamodel in the MDA TS. The appropriate mapping rules are generated by traversing the metamodel via its composite aggregation (in short composition) relationships.

$_{pAA}$: $MT_{mapAA} \rightarrow id$. In our example the model transformation relations identified by the bootstrapping are $MT_{mapAA}\{Process \leftrightarrow Process, Task \leftrightarrow Task\}$.

- ① The inference engine derives relationships between N_A' and N_B in the Ontology TS. This is possible, since both N_A' and N_B are mapped to the same reference ontology N_{RO}. It is automatically computed, how the concepts of N_A' can be substituted by semantically identical concepts of N_B ($\sigma(MT_{mapAA})=MT_{mapAB}$). Those relationships can be transferred to the MDA TS as the modelling languages A and B have synchronous representations in both MDA TS and Ontology TS. The substitutions computed for our example are *[EPC/Process]* and *[EPCElement/Task]*.
- ② Finally, the concepts of N_A' are substituted with the concepts of N_B in the model of MT_{mapAA} and we obtain a model of the model transformation MT_{mapAB} with $M_{MTmapAB}$: $MT_{mapAB} \rightarrow id$. The substitution is performed via modification operations on the model of the model transformation MT_{mapAA} in MDA TS. In the example the following model transformation relations are generated: $MT_{mapAB}\{Process \leftrightarrow EPC, Task \leftrightarrow EPCElement\}$.

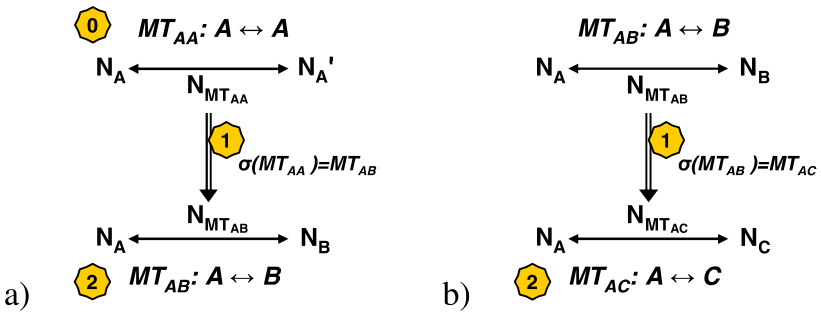

Fig. 4. Procedure of a) automated mapping generation, b) model transformation evolution

4.2 Evolution of Model Transformations

OntMT also fosters the evolution and reuse of existing model transformations. Instead of performing the bootstrapping step, the procedure for model transformation evolution takes the model transformation that shall be reused as input (see Figure 4b). This initial model transformation MT_{mapAB}: $A \leftrightarrow B$ encodes knowledge about how modelling language A is translated into B. The steps ① and ② are the same as for automated mapping generation. In step ①, a substitution $\sigma(MT_{mapAB})=MT_{mapAC}$ is computed on the basis of inference results. Step ② applies this substitution and generates a new version of the initial model transformation MT_{mapAC}: $A \leftrightarrow C$. The bootstrapping step helps to extend ontMT to scenarios where existing model transformations are adjusted. Avoiding to derive model transformations directly from ontologies results in a more flexible and well-structured architecture. OntMT can both generate new model transformations and reuse knowledge encoded in existing transformations. Issues concerning the model transformation, like checking if its model conforms to the QVT metamodel or considering the cardinality of associations' ends, are all dealt within the MDA TS. The Sem-MT-Component invokes modifications operations on the basis of the reasoning results and the application of heuristics.

5 Realization of OntMT

This section presents the components and concepts of ontMT realized as a tool for a semantic-enabled modelling and development suite, its parts and functionality.

5.1 Components of a Sem-MT-Tool

OntMT, as part of our vision of a semantic-enabled modelling and development suite, is realized as Sem-X-Tool (see Figure 5) [2]. The infrastructure provides basic functionality including a bridge ⓪ between models of the MDA TS and application ontologies of the Ontology TS (like it is described in [6]) and an inference component, which can be individually configured and used by Sem-X-Tools registered at the infrastructure. Sem-X-Tools, like the Sem-MT-Tool presented in this paper, are built on top of the infrastructure. They consist of a model manipulator, a Sem-X-Component, and a rule set. The model manipulator reads, creates, modifies and deletes models of the model repository ①. It delivers information about models to the Sem-X-Component ② and provides interfaces for model manipulation ③. The Sem-X-Component implements the core functionality of a Sem-X-Tool. It makes use of the reasoning results gained by inferring ontologies and computes the respective model manipulation ④. Since Sem-X-Tools are based on different relationships between the ontologies' elements, each Sem-X-Tool has its own set of reasoning rules.

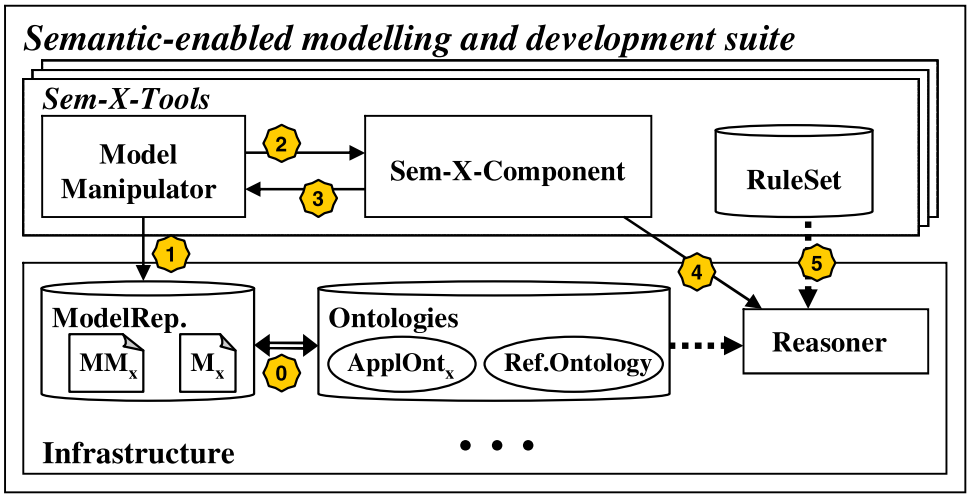

Fig. 5. OntMT as part of a semantic-enabled modelling and development suite

Figure 6 shows the architecture of the components building the Sem-MT-Tool which is an instantiation of the Sem-X-Tool. The model manipulator provides functionality via three interfaces: one that identifies the concepts of a metamodel that have to be substituted in a model transformation, one that performs a substitution of a metamodel's concepts in the model transformation, and one that provides validation functionality for the generated model transformation. The inference component provides an interface for accessing the reasoning results, i.e. the relationships between the metamodel elements. The Sem-MT-Component is the component of the Sem-MT-Tool, which connects the inference results of the Ontology TS to concrete modification actions on the models of the model transformation in the MDA TS.

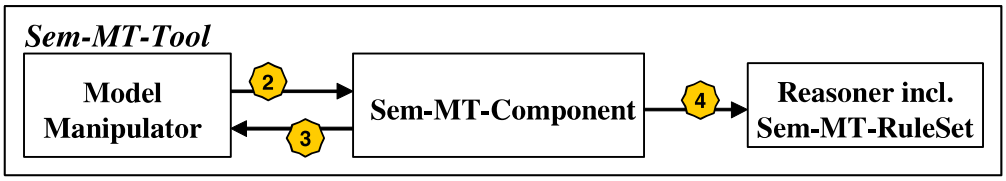

Fig. 6. Sem-MT-Tool component architecture

5.2 Architecture of OntMT

To validate our approach we have implemented a prototype that realizes the crucial parts of ontology-based model transformation. The following section provides more details about the architecture and the implementation. Therefore the architectural figures additionally depict the technologies we used to implement our prototype.

5.2.1 Inference Component

Figure 7 depicts a detailed architectural view on the inference component of ontMT. The inference component consists of a knowledge base and a reasoner. The base graph contains all facts of the knowledge base before the reasoning, i.e. the reference ontology, application ontologies[5], and the ontological groundings. The reasoner is triggered by rules specific to the Sem-MT-Tool, and computes the inference graph on the basis of the base graph. As the result of the reasoning, the knowledge base contains information about all relationships that are important for ontMT. These are the relationships between the application ontologies.

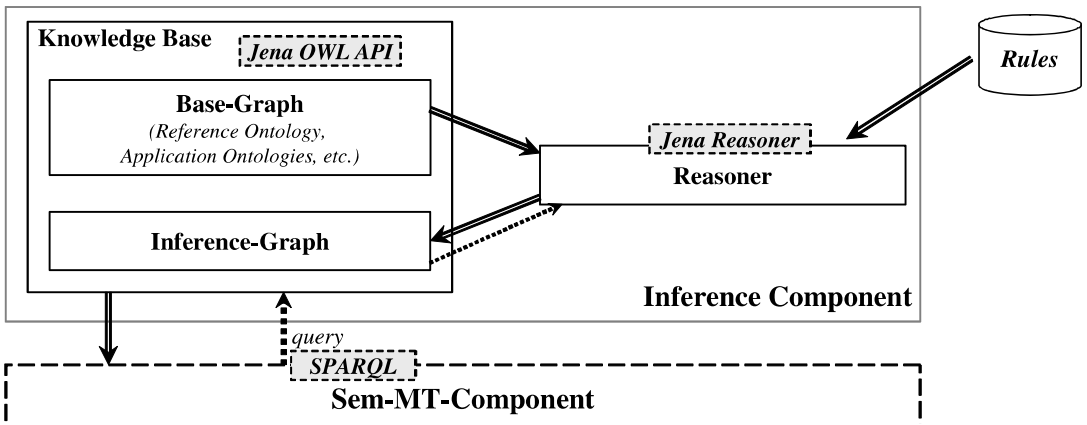

Fig. 7. Inference component

In [9], [29], and [40], *equivalence*, *containment*, and *overlap* are described as the main relationships for mapping ontologies. The inference component identifies (for ontMT) these relationships between the ontology elements. The relationships are also used for the ontological groundings by specifying mappings between the application ontologies and reference ontologies. This is possible, since the model elements are represented in application ontologies via the UML to OWL mapping described in the ODM standard [35, p.201ff].

[5] An application ontology corresponds to a metamodel in the Ontology TS.

- *"Equivalence"* (≡) means that the connected elements represent the same aspect of the real world. An element of an application ontology corresponds to an element in the reference ontology or can be precisely expressed by a composition of elements. Later, we will refer to this relationship by the relationship type <*equal*>.
- *"Containment"* (A,B) states that the element in one ontology represents a more specific aspect of the world than the element in the other ontology. The relationship can be defined in one or the other direction, depending on which concept is more specific. When an element is not sufficiently refined, i.e. it does not match the accuracy level of the ontology, we use the relationship <*general*>. When an element is described at a level of refinement that does not match the level of refinement of the other ontology we use the relationship <*special*>.
- *"Overlap"* (*o*) states that the connected elements represent different aspects of the world, but have an overlap in some respect. This relationship is of the type <*overlap*>.

Implementation
In our current prototype we use the Jena ontology API[6] to create and handle ontologies. The inference is realized through rules deployed to the rule engine included in Jena. Jena also uses this rule engine to provide (partial) RDFS and OWL reasoning[7]. The rule 2 in Listing 2 for example states, that if A overlaps B and B is an intersection of C and D then A overlaps C and D. The inference results are obtained with SPARQL, which queries the knowledge base for the relationships between the application ontologies.

rule 1: $A \; o \; B \wedge B \sqsubseteq C \rightarrow A \; o \; C$
rule 2: $A \; o \; B \wedge B \equiv C \sqcap D \rightarrow A \; o \; C \wedge A \; o \; D$

Listing 2: Sample reasoning rules

The decision to use the Jena framework and its rule based reasoning support for the prototype implementation was mainly based on two decisions. First, it better met our requirements, which were mainly a combination of TBox reasoning, rule support, and good documentation, than other open source projects. Second, the Jena framework provides the possibility to integrate other reasoners like Pellet[8] or future implementations of ontology mapping approaches using local domains like C-OWL [8].

5.2.2 Model Manipulator
The model manipulator provides modification operations on model transformations. It implements a language for model transformation modification that is used by the Sem-MT-Component to trigger the modification of the model transformations via modification programs. The semantics of this model transformation modification language treats model transformations as models. The fact that model transformation languages like QVT are represented through metamodels and modeltransformation programs are

[6] http://jena.sourceforge.net/tutorial/RDF_API/
[7] http://jena.sourceforge.net/inference/
[8] http://pellet.owldl.com/

models allows higher-order transformations, like transformations taking other transformations as input and producing transformations as output [4].

Due to the gap between the concepts of DSLs and metamodels implementing these DSLs, the semantics of the model transformation modification language needs to provide mechanisms to allow the Sem-MT-Component to adapt a modification program to the best possible solution. Hence the semantics is divided into a *modification semantics* and a *checking semantics* (see Figure 8).

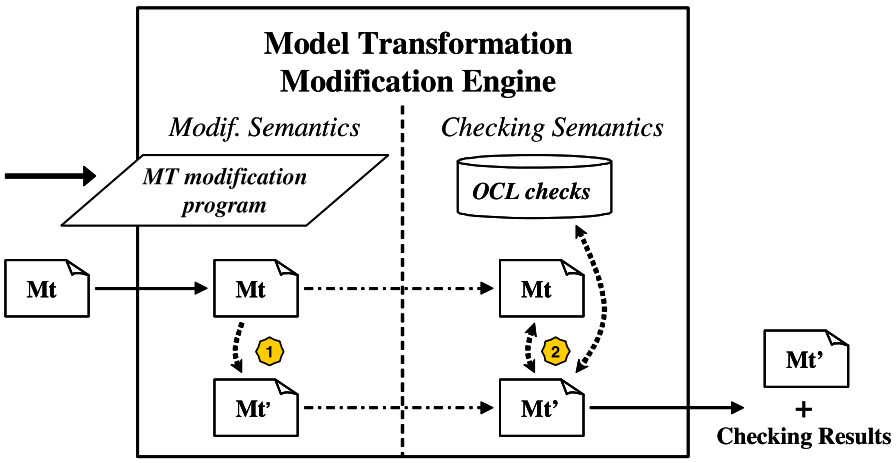

Fig. 8. Semantics of model transformation modification

Modification semantics

The modification semantics defines how the modification of model transformations, which is specified in modification programs, is executed. A simplified picture that helps to work with the model transformation modification language is to imagine the modification program as a substitution. The elements of the modification program's source metamodel are substituted with the elements of the target metamodel. The detailed implementation realizing the semantics is encapsulated in a separate component of the model manipulator. Currently realized substitution operators provide functionality for one-to-one, one-to-many, and removal substitutions of both classes and properties. In the following we give a short outline of the substitution operators' functionality via short examples.

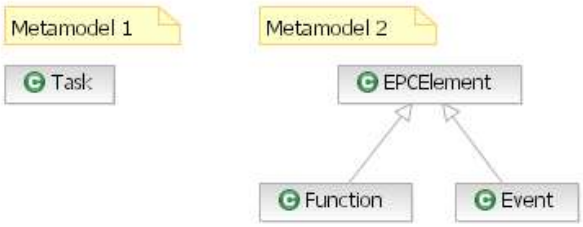

Fig. 9. Two example metamodels

```
relation rule {
  checkonly domain l_mm var1:Task { };
  enforce domain r_mm var1':Task { };
}
```

Listing 3: Example model transformation specification (notation similar to QVT)

- **One-to-one substitution:** If the *Task* in the sample model transformation rule shall be substituted by *EPCElement* in the right-hand model, then the one-to-one substitution for classes has to be applied: [*EPCElement*/*Task*]. The result of applying this one-to-one substitution to the transformation rule of Listing 3 is as follows:

```
relation rule' {
  checkonly domain l_mm var1:Task { };
  enforce domain r_mm var1':EPCElement { };
}
```

Listing 4: Model transformation after applying one-to-one substitution

- **One-to-many substitution:** If the *Task* in the sample model transformation rule (see Listing 3) shall be substituted by *Function* and *Event*, then the one-to-many substitution for classes has to be applied: [{*Function*,*Event*}/*Task*]. The result of the one-to-many substitution is not so obvious like the result of the one-to-one substitution since the model transformation rule has to be duplicated. Details about one-to-many substitutions can be found in the case study in Section 6.

```
relation rule'_a {
  checkonly domain l_mm var1:Task { };
  enforce domain r_mm var1':Function { };
}

relation rule'_b {
  checkonly domain l_mm var1:Task { };
  enforce domain r_mm var1':Event { };
}
```

Listing 5: Model transformation after applying one-to-many substitution

- **Removal substitution:** A removal substitution is sensibly applied when an element of the source metamodel cannot be substituted by any element of the target metamodel. If e.g. a removal substitution [-/*Task*] is applied to the *Task*, the whole transformation rule of Listing 3 would be removed from the model transformation.

Checking semantics

The checking semantics represents the second part of the model transformation modification language's semantics. It tests the generated model transformations for so-called problems, which can occur by applying the modification semantics. One set of problems affects the consistency of model transformation programs with respect to the model transformation language, i.e. the generated model transformations are not valid and cannot be executed. Another kind of problems is caused, when knowledge encoded into the original model transformation is not preserved or lost. This is the case when modifications and substitutions are applied to relations where they (normally) do not make sense. To detect the second kind of problems the generated model transformation has to be compared with the original model transformation. In general, problems are detected via OCL [34] constraints. Only for a few problems, where additional information about the execution of the modification is needed, we extend this mechanism with information from the modification execution. The following list describes the main problem types and provides some sample OCL constraints:

- *The substitution of property failed (PropertySubstitutionFailed):* This problem occurs, when the model transformation modification program did not specify a substitution for a property that is used in the model transformation. Since in such cases the modification semantics sets the value of the property to *UNDEFINED*, the OCL constraint in Listing 6 checks whether the value of the property (*referredProperty*) is not empty.

```
package qvttemplate
  context PropertyTemplateItem
    inv referredProperty_must_be_set: self.referredProperty→notEmpty()
endpackage
```

Listing 6: OCL constraint: substitution of property failed

- *The substitution of class failed (ClassSubstitutionFailed):* This problem occurs, when the model transformation modification program did not specify a substitution for a class that is used in the model transformation.
- *A property is not part of class (PropertyNotPartOfClass):* The generated model transformation would require a property to be part of a class, what is not the case in the respective metamodel (model types as described in [41] do not match). The OCL invariant *property_part_of_class* in Listing 7 is used to check this fact. The constraint is satisfied, if the OTE's referred class (cp. QVT relational description in Section 2) contains the property or if the OTE does not reference a class. The second case occurs when the class of the OTE could not be substituted.

```
package qvttemplate
  context PropertyTemplateItem
    inv property_part_of_class:
      self.objContainer.referredClass.hasProperty(self.referredProperty)
      or self.objContainer.referredClass→isEmpty()
endpackage

package emof
  context Class
  def: hasProperty(property: Property): Boolean =
    self.ownedAttribute→exists(p: Property | p = property)
    or self.superClass→exists(c: Class | c.hasProperty(property))
endpackage
```

Listing 7: OCL constraint: property part of class

- *A property has not the type of class (PropertyNotTypeOfClass):* This constraint checks, whether a property of the generated model transformation has only types in the model transformation that are compatible with the types of the respective metamodel. The constraint is not satisfied, when the types of the properties inferred from the model transformation and the metamodel do not match (model types as described in [41] do not match).
- *Granularity of initial model transformation is not appropriate (InitialMTGranularityNotAppropriate):* This problem occurs when a one-to-many substitution has to be applied to top level relations. Since the granularity of the initial model transformation does not match the level of refinement of the new metamodel, i.e. the

model transformation relations are too coarse grained, the substitution operators (especially one-to-many) cannot be sensibly applied.[9]

- *Pattern removal (PatternRemoval):* A loss of information occurs when a class or a property could not be substituted and therefore the respective patterns had to be removed from the model transformation.

Architecture and Implementation

The model manipulator component is divided in a front and a back end (see Figure 10). The front end primarily conducts tasks that depend on the source language, while the back end deals with all issues specific to the target language. The metamodels and the bootstrap model transformation are brought into an intermediate representation format by the scanner and the parser. The substitution algorithm performs the substitutions proposed by the Sem-MT-Component. The validator checks whether any performed substitution leads to problems in the new model transformation.

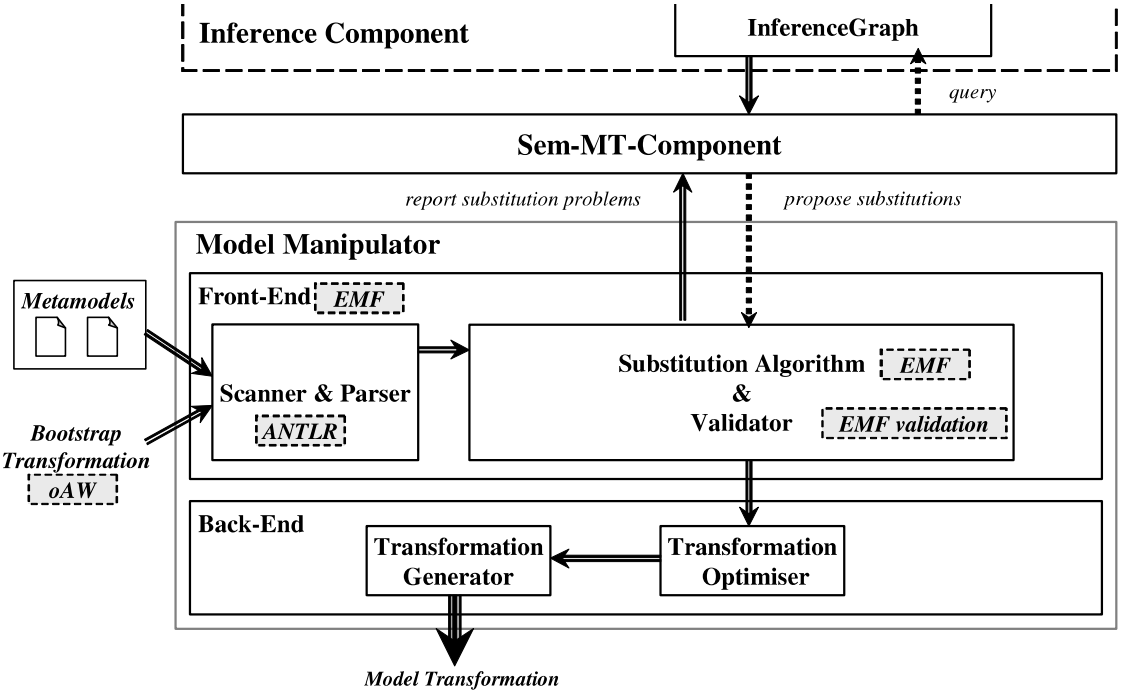

Fig. 10. Model manipulator

Our prototypical implementation of the model manipulator is based on Eclipse. It uses the Eclipse Modeling Framework (EMF). EMF allows the model manipulator to treat the metamodels and the model transformations with a single model manipulation API. This reflective API allows to handle EMF objects regardless of which metamodel they are (EMOF, QVT, OCL, etc.) generically. The metamodels are instantiations of the EMF EMOF implementation and the model transformation models are treated as instantiations of the EMF QVT relational implementation. Since the first final adopted version of the QVT standard [36] contains some inconsitencies we had to make some adjustments which are documented in our implementation.

[9] As we can see in the case study example in Section 6, an automatic bootstrapping algorithm can avoid this problem, if the refinement levels of the metamodels do match.

- Parser: The implementation of the parser makes use of the ANTLR parser generator [38] and parses a QVT relational textual syntax into EMF QVT relational models. It has been made available under GPL via the QVT Parser project[10].
- A prototype of the model manipulator implementation is part of the OntMT project[11]. The substitution algorithm is totally based on the EMF API. The validation component uses the EMF validation framework to check the EMF model of the generated QVT relational transformation with OCL constraints. Since we use the Eclipse Modeling Framework (EMF) [11], the Eclipse Validation Framework [12] is a consequent choice for implementing and performing the OCL checks. The results of EMF validation lend themselves very well to determine the exact position of problems or inconsistencies in a model transformation. OCL constraints, checking whether a model transformation is syntactically correct, can be automatically generated from the QVT metamodel. It is checked whether the model transformation conforms to the grammar that the QVT metamodel was generated from. With further manually implemented OCL constraints the model manipulator checks whether the generated model transformation is valid and can be executed or whether knowledge has been lost through the substitution.
- The bootstrapping generates from a metamodel expressed in MOF a QVT relational model transformation. It is implemented with templates expressed in the oAW[12] expand-language and available via the OntMT project. The bootstrapping is well integrated in the model manipulator, since EMF models can be used for oAW code generation. In fact the same metamodels that are used by the QVT-Parser and the model manipulator are also used for the bootstrapping.

5.2.3 Sem-MT-Component

The Sem-MT-Component implements the core part of the ontMT approach. It provides the main functionality of the Sem-MT-Tool. It makes use of the inference results of the Ontology TS and computes modifications programs for the generation and evolution of model transformations in the MDA TS. Listing 8 illustrates the algorithm implemented by the Sem-MT-Component to generate the new model transformations.

The Sem-MT-Component takes as input an initial model transformation, the metamodel which has to be substituted in the model transformation, and the new metamodels. In a first step (1) it requests the model manipulator to compute a set of all classes and properties that have to be substituted in the initial model transformation. Second, it invokes the inference component to obtain possible substitutions for all classes and properties to substitute (2). Next the computation of a substitution proposal begins. A substitution proposal contains the model transformation modification program, the problems that occur by applying the substitutions to a model transformation, and a rating of the performed substitutions. After a substitution proposal is calculated by the Sem-MT-Component (3), the model manipulator performs the substitution (4) and validates (5) the generated model transformation. Then the substitution proposal is rated by the Sem-MT-Component (6). The Sem-MT-Component tries to compute alternative substitution proposals until their application does not lead

[10] http://sourceforge.net/projects/qvtparser/
[11] http://sourceforge.net/projects/ontmt/
[12] http://www.eclipse.org/gmt/oaw/

to any problems, or no further substitution proposals can be found. Finally the Sem-MT-Component chooses the substitution proposal with the best rating from all computed substitution proposals (7) and the new model transformation is generated on the basis of this substitution proposal (8).

```
input:   initialMT: the initial model transformation
         subMM: metamodel to substitute in the model transformation
         newMM: new metamodel
output:  newMT: new generated model transformation

begin
    let sCS = set of concepts to substitute in the model transformation
    let sCRS = set of tuples with concepts to substitute in the model
               transformation and their possible substitutions
    let subProp = current substitution proposal
    let setSubProp = set of substitution proposals

(1) sCS := identifyConceptsToSubstitute(initialMT, subMM, newMM);

    foreach c ∈ sCS
(2)   sCRS := sCRS ∪ {(c, findPossibleSubstitutionsForConcept(c))};
    end foreach

    do
(3)   subProp := calculateSubstitutionProposal(cCRS);

(4)   tempMT := performSubstitution(subProp);

(5)   subProp := validateSubstitution(tempMT, subProp);

(6)   subProp := rateSubstitution (subProp);
    until subProp.problems == ∅ OR NoOtherSubProposalsPossible end do

(7)   subProp := chooseSubPropWithBestRating(setSubProp);
(8)   newMT := performSubstitution(subProp);

    return newMT
end
```

Listing 8: Algorithm to compute new model transformation (Sem-MT-Component)

Rating substitutions proposals

The choice of the substitution proposal, which is used to generate the new model transformation, is based on the ratings of the substitution proposals. A rating of a substitution proposal is a measure of the generated model transformation's quality. The rating is based on factors that are measured for each substitution proposal:

- **Problems occurring in the substitution proposal**: This measure counts the problems that were detected by the validator in the generated model transformation. The measure distinguishes between the different kinds of problems and assigns different weights to the various problem types according to the severity.
- **Number of concepts that could be substituted**: This measure counts how many class and properties could be substituted. From this measure can be derived how many concepts could not be substituted.

- **Relationships used for substitution**: This measure counts and rates the relationships that are used in the modification program of the substitution proposal. In general a substitution derived from an <*equal*> relationship gets a better rating than substitutions derived from other relationships.
- **Position of the problems and used relationships in the model transformation**: This is an optional factor that can influence the three other ratings describe above. The assumption for this factor is, that some relations of the model transformation are more important to the overall result than others.

Concrete ratings depend always on the purpose ontMT is used for. For the different application scenarios separate metrics are defined. A metric, which was developed for automated mapping generation, will put more emphasis on an executable model transformation than on the relationships used for substitution. If ontMT is used to support developers in adjusting their model transformations, ontMT will only make suggestions to the developer. Hence, the metric puts more emphasis on exact substitutions of metamodel elements than on the execution of the new model transformation.

Implementation
The OntMT project currently provides a simple implementation of the correlation algorithm, which is described in Listing 8. However, an automated synchronisation of the modelling and the reasoning world (see © in Figure 5) is not yet fully integrated. We are developing a prototype that synchronizes EMF and Jena OWL models and allows to answer SPARQL like queries on EMF models with reasoning support. The synchronization mechanism makes use of the UML to OWL mapping described in the ODM standard [35, p.201ff]. However, we plan to replace our prototype with an implementation of the Eclipse EODM project[13]. This projects aims to provide inference capabilities for OWL models implemented in EMF and model transformations of RDF/OWL to other modelling languages such as UML.

6 Case Study about Automated Mapping Generation

This section provides further insights about how the Sem-MT-Tool works. It illustrates the automated mapping generation application scenario of ontMT that has been introduced in Section 4.1. A mapping between two metamodels (Figure 11 and 12) for process modelling is generated. The first metamodel *Process* is an excerpt of a metamodel for process orchestration in a service-oriented environment. The second metamodel *EPC* is also for process modelling.[14]

The reference ontology in this example (see Figure 13) is an excerpt of the Web Ontology Language for Services (OWL-S). For the ontological grounding we use a notation similar to SMAIL [29] (Semantic Mediation and Application Interoperability Language). '=:' stands for a lossless annotation, where the annotation fully captures the intended meaning. '>:' denotes an overspecification, where the level of refinement

[13] http://www.eclipse.org/modeling/mdt/?project=eodm
[14] The QVT model transformations of this case study together with the EMOF metamodels, and a few test models have been executed with ModelMorf [31] beta version 2.

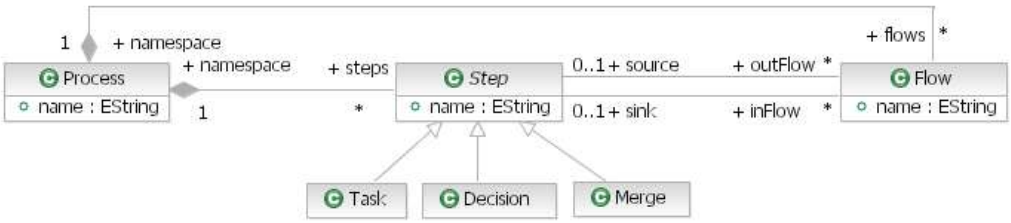

Fig. 11. Metamodel *Process*

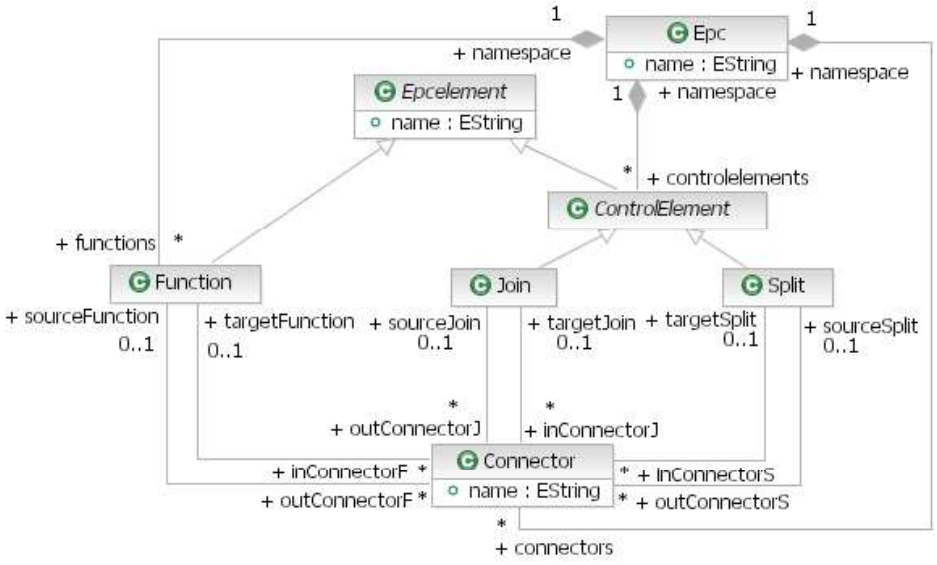

Fig. 12. Metamodel *EPC*

of the annotated element is greater than the level of refinement of the concepts in the reference ontology.

- In a first step, the bootstrapping generates the initial model transformation by traversing metamodel *MM1* via its composition (see Listing 9). The bootstrapping works as follows:
- For each class in the *Process* metamodel one *top relation* mapping rule is generated for the bootstrapping model transformation (*Process*, *Flow*). If there occurs inheritance, mapping rules for the concrete leaf classes (*Task*, *Decision*, *Merge*) are generated instead of mapping the abstract superclass (*Step*). This enhances the granularity of the model transformation specification. The mandatory properties are specified as part of the top relation (in the example the *name* property). Optional properties would be outsourced to separate relations, which are used to further constrain the *top relations* via *where*-statements.
- Composition associations are realized in the initial model transformation as properties of the contained elements (*namespace*). These properties constrain the *top relations* via *when*-statements (e.g. in the *FlowToFlow* relation).
- Other associations are realized via separate *relations* in the initial model transformation (*StepToStep_out*, *StepToStep_in*).

Table 1 and 2. Ontological Grounding of MM1 and MM2

Process [name,steps,flows]	=:	CompositeProcess [name,composedOf,composedOf]
Step [name,outFlow,inFlow,namespace]	=:	ProcessComponent [name,connected,inverseOf(followed), inverseOf(composedOf)]
Task [name,outflow,inFlow,namespace]	=:	AtomicProcess [name,connected,inverseOf(followed), inverseOf(composedOf)]
Flow [sink,source]	=:	FollowedBy [followed,inverseOf(connected)]
…	…	…
EPC [name,connectors]	=:	CompositeProcess [name,composedOf]
EPC [name,functions,connectors]	>:	CompositeProcess [name,composedOf,composedOf]
EPC [name,controlelements,connectors]	>:	CompositeProcess [name,composedOf,composedOf]
EPCElement [name]	=:	ProcessComponent [name]
Function [name,outConnectorF,inConnectorF, namespace]	=:	AtomicProcess [name,connected,inverseOf(followed), inverseOf(composedOf)]
Connector [sinkFunction,soureFunction, sinkJoin,sourceJoin, sinkSplit,souceSplit]	>:	FollowedBy [followed,inverseOf(connected), followed,inverseOf(connected), followed,inverseOf(connected)]
…	…	…

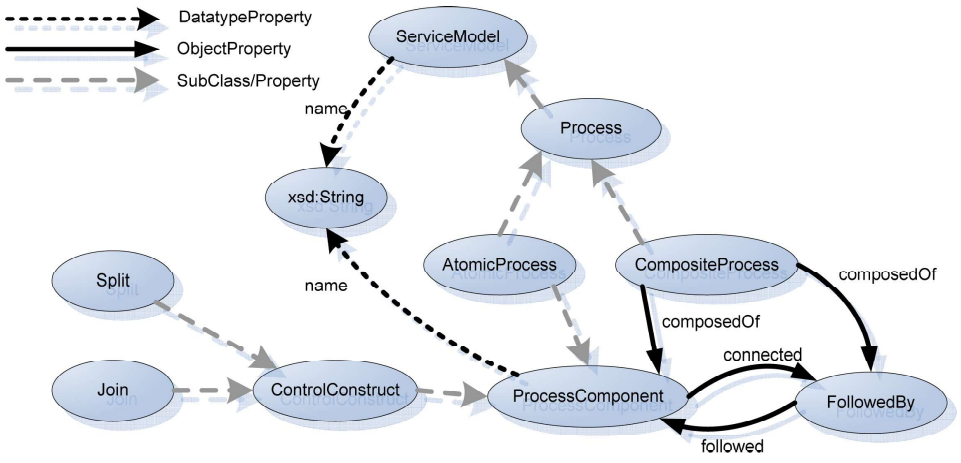

Fig. 13. Reference Ontology

- The model of the initial model transformation serves as input for the model manipulator. The first task of the model manipulator is to determine the classes and properties of the right-hand metamodel, which have to be substituted in the relations of the input model transformation.

```
transformation ProcessToProcess(prc_1:processMM_1; prc_2:processMM_2) {

key processMM_2::Process {name};
key processMM_2::Task {name, namespace};
key processMM_2::Decision {name, namespace};
key processMM_2::Merge {name, namespace};
key processMM_2::Flow {name, namespace};

top relation ProcessToProcess {
  pn: String;
  checkonly domain prc_1 p_1:Process { name=pn };
  enforce domain prc_2 p_2:Process { name=pn };
}

top relation TaskToTask {
  tn: String;
  checkonly domain prc_1 t_1:Task { namespace=p_1:Process {}, name=tn };
  enforce domain prc_2 t_2:Task { namespace=p_2:Process {}, name=tn };
  when { ProcessToProcess(p_1, p_2); }
  where {  StepToStep_out(t_1, t_2); StepToStep_in(t_1, t_2); }
}

top relation DecisionToDecision {
  dn: String;
  checkonly domain prc_1 d_1:Decision { namespace=p_1:Process {},
                                        name=dn };
  enforce domain prc_2 d_2:Decision {namespace=p_2:Process {}, name=dn};
  when { ProcessToProcess(p_1, p_2); }
  where {  StepToStep_out(d_1, d_2); StepToStep_in(d_1, d_2); }
}

top relation MergeToMerge {
  mn: String;
  checkonly domain prc_1 m_1:Merge { namespace=p_1:Process {}, name=mn};
  enforce domain prc_2 m_2:Merge { namespace=p_2:Process {}, name=mn};
  when { ProcessToProcess(p_1, p_2); }
  where {  StepToStep_out(m_1, m_2); StepToStep_in(m_1, m_2); }
}

relation StepToStep_out {
  fn: String;
  checkonly domain prc_1 s_1:Step { outFlow=out_1:Flow { name=fn } };
  enforce domain prc_2 s_2:Step { outFlow=out_2:Flow { name=fn } };
}

relation StepToStep_in {
  fn: String;
  checkonly domain prc_1 s_1:Step { inFlow=in_1:Flow { name=fn } };
  enforce domain prc_2 s_2:Step { inFlow=in_2:Flow { name=fn } };
}

top relation FlowToFlow {
  fn: String;
  checkonly domain prc_1 f_1:Flow { namespace=p_1:Process {}, name=fn };
  enforce domain prc_2 f_2:Flow { namespace=p_2:Process {}, name=fn };
  when { ProcessToProcess(p_1, p_2); }
}
}
```

Listing 9: The initial model transformation in QVT relational syntax

- For each concept to substitute the Sem-MT-Component queries the inference component for relationships. The inference component searches the knowledge base for triples like <*Step:Process*> <*?*> <*?:EPC*>[15]. The result for <*Step:Process*> is:

  <*Step:Process* > <*equal*> <*EPCElement:EPC*>
  <*Step:Process* > <*general*> <*Function:EPC*>
  <*Step:Process* > <*general*> <*ControlElement:EPC*>
  <*Step:Process* > <*general*> <*Split:EPC*>
  <*Step:Process* > <*general*> <*Join:EPC*>

With this input (metamodels, reference ontology, ontological groundings and initial transformation and possible substitutions) the computation of the substitution of N_A' with N_B can start:

- In the first substitution proposal, the Sem-MT-Component considers only facts with the predicate <*equal*>, in order to find the best possible substitution. Since for the ObjectProperty <*outFlow(Flow):Process*> and <*inFlow(Flow):Process*> no substitution in the context of <*Epcelement:EPC*> is possible, this ObjectProperty is omitted in the substitution, in the hope that this does not affect the model transformation. Thus the substitutions of the first substitution proposal are:

Table 3. Substitution proposal *SP1*

Process[name]	→	EPC[name]
Step[outFlow,inFlow]	→	Epcelement[---,---]
Task[name,namespace]	→	Function[name,namespace]
Decision[name,namespace]	→	Join[name,namespace]
Merge[name,namespace]	→	Split[name,namespace]
Flow[name,namespace]	→	Connector[name,namespace]

The model manipulator generates a new model transformation on the basis of the first substitution proposal.

```
transformation ProcessToEpc_v1(prc_1:processMM_1; epc_1:epcMM_1) {

...

top relation TaskToFunction {
  tn: String;
  checkonly domain prc_1 t_1:Task { namespace=p_1:Process {}, name=tn };
  enforce domain epc_1 f_1:Function { namespace=e_1:Epc {}, name=tn };
  when {
    ProcessToEpc(p_1, e_1);
  }
  where {
    StepToEpcelement_out(t_1, f_1);
    StepToEpcelement_in(t_1, f_1);
  }
}

...
```

[15] The facts of in the knowledge base are of the form <*subject*> <*predicate*> <*object*>.

```
relation StepToEpcelement_out {
  fn: String;
  checkonly domain prc_1 s_1:Step { outFlow=out_1:Flow { name=fn } };
  enforce domain epc_1 e_1:Epcelement { };
}

relation StepToEpcelement_in {
  fn: String;
  checkonly domain prc_1 s_1:Step { inFlow=in_1:Flow { name=fn } };
  enforce domain epc_1 e_1:Epcelement { };
}

...

}
```

Listing 10: Model transformation generated from substitution proposal *SP1*

- The model transformation is validated by the model manipulator, which detects two *PropertySubstitutionFailed* problems for the *inFlow* and *outFlow* properties of the class *Epcelement*.
- Thereon the Sem-MT-Component searches for an alternate substitution, which also considering facts with predicates other than <*equal*>. For a substitution decision it applies a hierarchy, where the predicate <*equal*> is better than <*special*> and <*special*> is better than <*general*>. The facts provided by the inference component are:

  <*outFlow:Process*> <*general*> <*outConnectorF:EPC*>
  <*inFlow:Process*> <*general*> <*inConnectorF:EPC*>
  <*outFlow:Process*> <*general*> <*outConnectorJ:EPC*>
  <*inFlow:Process*> <*general*> <*inConnectorJ:EPC*>
  <*outFlow:Process*> <*general*> <*outConnectorS:EPC*>
  <*inFlow:Process*> <*general*> <*inConnectorS:EPC*>

  Based on its history of previously proposed substitutions[16] and the fact, that no facts with the predicates <*equals*> or <*special*> exist, the Sem-MT-Component computes a new substitution proposal *SP2*. This substitution proposal proposes to substitute the *outFlow* property with the three different *outConnector* properties:

Table 4. Substitution proposal *SP2*

Process[name]	→	EPC[name]
Step[outFlow,inFlow]	→	Epcelement [outConnectorF&outConnectorJ&outConnectorS, inConnectorF&inConnectorJ&inConnectorS]
Task[name,namespace]	→	Function[name,namespace]
Decision[name,namespace]	→	Join[name,namespace]
Merge[name,namespace]	→	Split[name,namespace]
Flow[name,namespace]	→	Connector[name,namespace]

[16] The Sem-MT-Component has a history of its previous substitution proposals, so that it will not make the same proposal a second time and the search for substitutions terminates.

- The model manipulator generates a model transformation from the new substitution proposal:

```
transformation ProcessToEpc_v1(prc_1:processMM_1; epc_1:epcMM_1) {

...

relation StepToEpcelement_out_F {
  fn: String;
  checkonly domain prc_1 s_1:Step { outFlow=out_1:Flow { name=fn } };
  enforce domain epc_1 e_1:Epcelement {
                           outConnectorF=out_2:Connector { name=fn } };
}

relation StepToEpcelement_in_F {
  fn: String;
  checkonly domain prc_1 s_1:Step { inFlow=in_1:Flow { name=fn } };
  enforce domain epc_1 e_1:Epcelement {
                           inConnectorF=in_2:Connector { name=fn } };
}

relation StepToEpcelement_out_S {
  fn: String;
  checkonly domain prc_1 s_1:Step { outFlow=out_1:Flow { name=fn } };
  enforce domain epc_1 e_1:Epcelement {
                           outConnectorS=out_2:Connector { name=fn } };
}
...
}
```

Listing 11: Model transformation generated from substitution proposal *SP2*

The model transformation is validated by the model manipulator, which detects six *PropertyNotPartOfClass* problems, since e.g. the property *outConnector* is part of the class *Function* and not part of the class *Epcelement*.

- Thus the Sem-MT-Component calculates an alternative substitution proposal *SP3*, where a *Step* is substituted by *Function*, *Join*, and *Split*:

Table 5. Substitution proposal *SP3*

Process[name]	→	EPC[name]
Step[outFlow,inFlow]	→	Function[outConnectorF,outConnectorJ] Split[outConnectorS,inConnectorF] Join[inConnectorJ,inConnectorS]
Task[name,namespace]	→	Function[name,namespace]
Decision[name,namespace]	→	Join[name,namespace]
Merge[name,namespace]	→	Split[name,namespace]
Flow[name,namespace]	→	Connector[name,namespace]

- The model transformation generated on the basis of *SP3* is as follows:

```
transformation ProcessToEpc_Trans(prc_1:processMM_1; epc_1:epcMM_1) {

key epcMM_1::EPC {name};
key epcMM_1::Function {name, namespace};
```

```
key epcMM_1::Split {name, namespace};
key epcMM_1::Join {name, namespace};
key epcMM_1::Connector {name, namespace};

top relation ProcessToEpc {
  pn: String;
  checkonly domain prc_1 p_1:Process { name=pn };
  enforce domain epc_1 e_1:Epc { name=pn };
}

top relation TaskToFunction {
  tn: String;
  checkonly domain prc_1 t_1:Task { namespace=p_1:Process {}, name=tn };
  enforce domain epc_1 f_1:Function { namespace=e_1:Epc {}, name=tn };
  when { ProcessToEpc(p_1, e_1); }
  where {
    StepToEpcelement_out_F(t_1, f_1);
    StepToEpcelement_in_F(t_1, f_1);
  }
}

...

relation StepToEpcelement_out_F {
  fn: String;
  checkonly domain prc_1 s_1:Step { outFlow=out_1:Flow { name=fn } };
  enforce domain epc_1 e_1:Function {
                          outConnectorF=out_2:Connector { name=fn } };
  when { FlowToConnector(out_1, out_2); }
}

relation StepToEpcelement_in_F {
  fn: String;
  checkonly domain prc_1 s_1:Step { inFlow=in_1:Flow { name=fn } };
  enforce domain epc_1 e_1:Function {
                          inConnectorF=in_2:Connector { name=fn } };
  when { FlowToConnector(in_1, in_2); }
}

...

top relation FlowToConnector {
  fn: String;
  checkonly domain prc_1 f_1:Flow { namespace=p_1:Process {}, name=fn };
  enforce domain epc_1 c_1:Connector { namespace=e_1:Epc {}, name=fn };
  when { ProcessToEpc(p_1, e_1); }
}
}
```

Listing 12: Model transformation generated from substitution proposal *SP3*

- The validator of the model manipulator comes to the result, that this substitution proposal leads to a new model transformation in which none of the problems mentioned above occur.
- Thus the Sem-MT-Component stops computing new substitution proposals and compares the ratings of the substitution proposals already tested:

Table 6. Rating of the substitution proposals

Substitution Proposal	***Problems occurred***	***Substituted Concepts***	***Used Relationships***
SP1	2x *PropertySubstitutionFailed*	5x class 8x property	13x <*equal*>
SP2	6x *PropertyNotPartOfClass*	5x class 10x property	13x <*equal*> 6x <*general*>
SP3	---	5x class 10x property	12x <*equal*> 9x <*general*>

- The Sem-MT-Component decides to use the substitution proposal *SP3* to generate the new model transformation. This is based on the consideration that a model transformation generated from *SP2* will not be able to be executed due to typing problems. Furthermore *SP3* can substitute more concepts of the original model transformation than *SP1*. The model transformation between metamodel N_A (*Process*) to N_B (*EPC*) generated by ontMT is listed in Listing 12.

7 Assessment of OntMT

This section discusses the ontMT approach with respect to its partical application, its limits, and possible weaknesses.

7.1 Application Areas

Ontology-based model transformation fosters the exchange of models and the evolution of model transformations. Model exchange scenarios are build on the generation of new model transformations, while model transformation evolution scenarios aim at reusing model transformations. As introduced in Section 2, one can distinguish between horizontal and vertical model transformations. Horizontal model transformations are mappings between models at a certain abstraction level, where no information is lost and no additional information is added. Vertical model transformations are refinements that add additional information to the generated model about e.g. architecture or platform. Thus, the target model of a refinement is more detailed than the source model.

	Mapping	***Refinement***
Model Exchange	autom.gen. (+ man.)	n/a
MT Evolution	autom.gen. / autom.mod.	autom.mod.

Fig. 14. Application of ontMT to model exchange and model transformation evolution

Figure 14 categorizes the support that ontMT can provide to the described application scenarios and the different types of model transformations.

- To exchange models between different DSLs, metamodels and modelling styles, ontMT is able to automatically generate mappings. However, the level of automation depends on how different the DSLs and their modelling approaches are. It may be necessary to provide additional mapping information through an initial model transformation, which cannot be inferred from the ontologies.
- OntMT supports the evolution and reuse of existing mappings. The new model transformation can be either generated from scratch or obtained through adjusting the existing mapping. The more individual features, which are different to the core structure of the metamodels, are encoded in existing mappings, the more preferable it is to adjust existing mappings. The generation of new mappings is better, if the new metamodel provides extensions to the old one or a new modelling style specifies a fundamentally different composition of modelling elements.
- For the evolution and reuse of refinements, ontMT provides the possibility of automated modification and adjustment of existing model transformations. Refinement model transformations cannot be generated without human interaction, since they contain individual knowledge about software architecture or the platform, e.g. patterns like broker, model-view-controller, etc..

7.2 Evaluation

The ontMT approach adjusts initial model transformations in order to generate or maintain model transformations. Since mapping knowledge is captured in bindings of the metamodels to the reference ontology, one could favour an approach that derives model transformation rules directly from these bindings. This may very well work for model exchange scenarios. However, in model transformation evolution scenarios the model transformation itself would have to be encoded in the bindings. In our opinion, it is better to encode this transformation knowledge in an initial model transformation, i.e. the model transformation to reuse.

The level of automation that ontMT can provide highly depends on how different metamodels, DSLs, and modelling approaches are. If for example two DSLs totally differ in their modelling approaches, their metamodel bindings will be two mostly unconnected sets of the reference ontology. OntMT does not add real transformation knowledge that changes the semantics of model transformation. It depends on the results that are inferable via the ontologies that are used to adjust the syntax of model transformations.

We also made scalability considerations and tests for ontMT in terms of memory requirements, runtime, and size of model transformations that can be processed. This was done for the three components of ontMT (see Section 5.1) separately. Memory requirements and runtime of the model manipulator rise linear to the number of rules a model transformation contains. We tested this with model transformations that contain up to 200 rules. In ontMT reasoning has only to be performed once at runtime. Its memory requirements and runtime depends on the size and the complexity of the ontologies. Since the application scenarios of ontMT do not have hard real-time constraints, we do not see problems in practice concerning memory requirements, runtime, and size of model transformations for the model manipulator and the inference component.

However, the Sem-MT-Component can be seen as the 'bottleneck' of the ontMT approach. This component has to combine the reasoning results to a sensible input for the model manipulator. For this combination the size of the solution space grows exponentially with the relationships that are inferred for each concept. The size of the solution space is c^n, where c is the number of concepts in a metamodel and n is the number of relationships inferred for each concept. We try to solve this problem by restricting the solution space. As exemplified in the case study we apply heuristics that first guess an 'ideal' solution and the try to solve problems locally in the solution space, i.e. where the problems in the generated model transformation were detected.

7.3 Discussion

The ontMT approach assumes the existence of an appropriate reference ontology. However, developing or agreeing on a reference ontology is a non-trivial task. For example there may exist different versions of (reference) ontologies, what would transfer the problem of heterogeneous models from the MDA TS to the ontology TS. In those cases techniques for matching and merging ontologies, like linguistic, schema-based, or probabilistic approaches, combined with human intervention have to be applied to obtain a suitable reference ontology. Ontology alignment, matching, and mapping approaches can be also very useful to discover and define bindings from the metamodels to the reference ontology. [28] describes an approach and a conceptual framework for mapping distributed ontologies. It can provide the basis for an interactive and incremental mapping process that is needed for developing the bindings in ontMT. In such a process the SKOS mapping vocabulary [42] could be used to specify mappings between concepts from different ontologies. For this vocabulary a search algorithm has been developed [17] that can discover potential candidates for substitutions in ontMT.

To provide ontological groundings and to find reference ontologies may require investing a lot of effort. Depending on the concrete application scenario, this effort may not be justifiable with the generation and evolution of model transformations. Developing or adjusting model transformations by hand may be cheaper. Hence, the goal is to reuse reference ontologies and ontological groundings with other applications that are part of a semantic-enabled modelling and development suite (see Section 5.1).

A totally automated solution may also have to cope with acceptance problems of software engineers. Software engineers will probably not be willing to give up overall control of model transformation to an automated tool, which makes its choice based on metrics and heuristics. Hence, the majority of application scenarios will be of such a form, that the Sem-MT-Tool makes suggestions with a change and problem history to the software engineer. The engineer has the possibility to accept, correct, or reject the suggestions.

8 Related Work

In [41], the authors introduce model typing as extension of object-oriented typing and propose an algorithm for checking the conformance of model types. It is presented,

how model typing permits more flexible reuse of model transformations across various metamodels while preserving type safety. This approach improves the reuse of model transformations whenever small changes to metamodels occur, like altering the cardinality of an association. In case of major change in models' representation formats or modelling guidelines model transformations still have to be modified manually. Furthermore automatic mapping generation is not provided.

The ATLAS Model Weaver (AMW) tool implements the model weaving approach introduced in [5]. It enables the representation of correspondences between models in so-called weaving models, from which model transformations can be generated. Model weaving aims to improve efficiency in the creation and maintenance of model transformations. Nevertheless, creating weaving links is not automatic and weaving models have to be adjusted whenever changes DSLs, metamodels or model guidelines of source and target models are made.

The work described in [13] presents an approach to semi-automate the development of transformations via weaving models of the model weaving approach. It describes an iterative and incremental procedure of weaving link generation, similarity calculation, and weaving link selection.

The Model-based Semantic Mapping Framework (Semaphore) [27] follows a similar approach as the ATLAS Model Weaver. By aiming to support mappings between domain models, it supports (graphically) specification of mappings between DSLs. These specifications are saved in mapping model, which for example can be used to generate code to transform the models. Like in AMW mappings have to be specified manually and have to be adjusted when ever changes to the model representations or modelling guidelines occur.

The ATHENA Semantic Suite provides tools for improving interoperability between organisations. In this approach e.g. XML Schemas can be annotated by a reference ontology and reasoning rules can be specified, so that reasoner can convert XML documents. The reasoner can be used as mediator transforming messages at runtime. This approach could be extended to modelling by transforming XML serialisations of models. The problem is that there would be no traceability of transformation executions between models. However, this is a key feature for MDD [16] and cross-organisational modelling. Since this is provided by model transformation languages it is also supported by our approach.

The ModelCVS project [23] provides a framework for semi-automatic generation of transformation programs. By explicitly representing the concepts modelling language in ontologies, the goal is to derive *bridgings* (transformations) between the original metamodels from the mapping between the ontologies. The approach focuses on mappings in order to foster tool interoperability [24]. For reusing existing (refinement) model transformations, knowledge about the transformation would have to be captured in the ontologies or ontology mappings.

9 Summary and Outlook

The approach of ontology-based model transformation provides technology that fosters interoperability in model exchange and the evolution of model transformations. It integrates ontologies in MDSD and makes use of the reasoning capabilities of the

Ontology TS. By automated generation of mappings it offers new possibilities for the integration of domain specific languages and 'legacy' models in a plug&play manner. This makes it easier for new organisations to join collaborations. OntMT also supports organisations evolving their modelling techniques like using new and more advanced versions of modelling languages. It yields more efficient reuse of model transformations and the knowledge that is captured in those transformations. Nevertheless, ontMT uses additional information, which has to be provided by the people developing metamodels and domain specific languages.

Future work to extend and improve ontMT is manifold. The model manipulator will be extended with more expressive substitution mechanisms, which for example allow more complex pattern matching. The metrics and heuristics of the Sem-MT-Component have to be tested and improved via more case studies and application scenarios from various domains. Other reasoners and implementations of ontology mapping approaches have to be integrated in the inference component. Finally, the ontMT approach has to provide or adopt methodologies, which support discovery and development of reference ontologies and bindings between metamodels and ontologies.

References

1. Bauer, B., Müller, J.P., Roser, S.: A Decentralized Broker Architecture for Collaborative Business Process Modelling and Enactment. In: 2nd International Conference on Interoperability of Enterprise Software and Applications (2006)
2. Bauer, B., Roser, S.: Semantic-enabled Software Engineering and Development. In: 1st International Workshop on Applications of Semantic Technologies. LNI (2006)
3. Benguria, G., Larrucea, X., Elvesæter, B., Neple, T., Beardsmore, A., Friess, M.: A platform-independent model for service-oriented architectures. In: 2nd International Conference on Interoperability of Enterprise Software and Applications (2006)
4. Bézivin, J.: On the unification power of models. Software and System Modeling 4(2), 171–188 (2005)
5. Bézivin, J., Jouault, F., Valduriez, P.: First Experiments with a ModelWeaver. In: OOPSLA & GPCE Workshop (2004)
6. Bézivin, J., Devedzic, V., Djuric, D., Favreau, J.M., Gasevic, D., Jouault, F.: An M3-Neutral Infrastructure for bridging model engineering and ontology engineering. In: 1st International Conference on Interoperability of Enterprise Software and Applications (2005)
7. Borgo, S., et al.: OntologyRoadMap. WonderWeb Deliverable D15 (2002), `http://wonderweb.semanticweb.org`
8. Bouquet, P., Giunchiglia, F., van Harmelen, F., Serafini, L., Stuckenschmidt, H.: C-OWL: Contextualizing Ontologies. In: Fensel, D., Sycara, K.P., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 164–179. Springer, Heidelberg (2003)
9. Brockmanns, S., Haase, P.: A Metamodel and UML Profile for Networked Ontologies. In: 2nd Workshop on Semantic Web Enabled Software Engineering (2006)
10. Elvesæter, B., Hahn, A., Berre, A.-J., Neple, T.: Towards an Interoperability Framework for Model-Driven Development of Software Systems. In: 1st International Conference on Interoperability of Enterprise Software and Applications (2005)
11. Eclipse Modeling Framework (EMF), `http://www.eclipse.org/modeling/emf/`

12. Eclipse Validation Framework (VF), http://www.eclipse.org/emft/projects/validation/
13. Fabro, M.D.D., Valduriez, P.: Semi-automatic model integration using matching transformations and weaving models. In: 22nd ACM Symposium on Applied Computing, Model Transformation Track (2007)
14. Favre, J.M.: Foundations of Meta-Pyramids: Languages vs. Metamodels, Episode II: Story of Thotus the Baboon (2004)
15. Frankel, D.S.: Model Driven Architecture – Applying MDA™ to Enterprise Computing. Wiley, Chichester (2003)
16. Gardner, T., Griffin, C., Koehler, J., Hauser, R.: A review of OMG MOF 2.0 QVT Submissions and Recommendations towards the final Standard. In: MetaModelling for MDA Workshop (2003)
17. Gasevic, D., Hatala, M.: Searching Web Resources Using Ontology Mappings. In: K-CAP 2005 Workshop on Integrating Ontologies (2005)
18. Greiner, U., et al.: Designing and implementing cross-organizational business processes - Description and Application of a Modeling Framework. In: 2nd International Conference on Interoperability of Enterprise Software and Applications (2006)
19. Guarino, N.: Understanding, Building, and Using Ontologies. International Journal of Human-Computer Studies 46(2-3), 293–310 (1997)
20. Harel, D., Rumpe, B.: Meaningful Modeling: What's the Semantics of "Semantics"? IEEE Computer 37(10), 64–72 (2004)
21. IDEAS: A Gap Analysis (2003), http://www.ideas-roapmap.net
22. IDEAS: The Vision for 2010 (2003), http://www.ideas-roapmap.net
23. Kappel, G., Kapsammer, E., Kargl, H., Kramler, G., Reiter, T., Retschitzegger, W., Schwinger, W., Wimmer, M.: Lifting Metamodels to Ontologies: A Step to the Semantic Integration of Modeling Languages. In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (eds.) MoDELS 2006. LNCS, vol. 4199, pp. 528–542. Springer, Heidelberg (2006)
24. Kappel, G., Kapsammer, E., Kargl, H., Kramler, G., Reiter, T., Retschitzegger, W., Schwinger, W., Wimmer, M.: On Model and Ontologies – A Layered Approach for Model-based Tool Integration. In: Proceedings of Modellierung (2006)
25. Klein, R., Kupsch, F., Scheer, A.-W.: Modellierung inter-organisationaler Prozesse mit Ereignisgesteuerten Prozessketten. In: Scheer, A.-W (Hrsg.): Veröffentlichungen des Instituts für Wirtschaftsinformatik, Nr. 178 (2004)
26. Kurtev, I., Bézivin, J., Aksit, M.: Technological Spaces: An Initial Appraisal. In: Int. Federated Conference (DOA, ODBASE, CoopIS) (2002)
27. Limyr, A., Neple, T., Berre, A.-J., Elvesæter, B.: Semaphore – A Model-Based Semantic Mapping Framework. In: Workshop on Enterprise and Networked Enterprises Interoperability at BPM 2006 (2006)
28. Maedche, A., Motik, B., Silva, N., Volz, R.: MAFRA - A Mapping Framework for Distributed Ontologies. In: 13th European Conference on Knowledge Engineering and Knowledge Management (2002)
29. Missikoff, M., et al.: A Controlled Language for Semantic Annotation and Interoperability in e-Business Applications. In: Workshop on Semantic Integration (2003)
30. MDDi - Eclipse project, http://www.eclipse.org/mddi/
31. ModelMorf, http://www.tcs-trddc.com/ModelMorf/index.htm
32. MODELWARE project, http://www.modelware-ist.org/
33. Oberle, D.: Semantic Management of Middleware. Springer, Heidelberg (2005)
34. OMG: Object Constraint Language, Version 2.0, formal/2006-05-01
35. OMG: Ontology Definition Metamodel, ad/2006-05-01

36. OMG: MOF QVT Final Adopted Specification, ptc/05-11-01
37. OMG: Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification - Final Adopted Specification, Final Adopted Specification, ptc/07-07-07
38. Parr, T.: ANTLR 3.0 (2007), http://antlr.org/
39. Roser, S., Bauer, B.: An Approach to Automatically Generated Model Transformation Using Ontology Engineering Space. In: 2nd Workshop on Semantic Web Enabled Software Engineering (2006)
40. Serafini, L., Stuckenschmidt, H., Wache, H.: A formal investigation of mapping language for terminological knowledge (2005)
41. Steel, J., Jézéquel, J.-M.: Model Typing for Improving Reuse in Model-Driven Engineering. In: Briand, L.C., Williams, C. (eds.) MoDELS 2005. LNCS, vol. 3713, pp. 84–96. Springer, Heidelberg (2005)
42. W3C: SKOS Mapping Vocabulary Specification, http://www.w3.org/2004/02/skos/mapping/spec/