WP1-D3
Report on transformation patterns

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-D3/D/PU/a1
Responsible editors:	Fernando Silva Parreiras, Tobias Walter, Christian Wende
Reviewers:	Krzysztof Miksa, Andreas Bartho
Contributing participants:	UoKL, CMR, UNIABDN, TUD
Contributing workpackages:	WP1, WP2, WP3, WP5
Contractual date of deliverable:	31 July 2009
Actual submission date:	17 August 2009

Abstract
Model Transformations are essential in order to support different kinds of models in a model driven environment. Model transformations are the main link between metamodelling technical spaces and ontology technical spaces. This deliverable describes frequently used patterns for transforming instantiations of metamodelling 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 in context of MOST.

Keyword List
software modelling, ontologies, transformation, composition, integration

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

Report on transformation patterns

Fernando Silva Parreiras[1], **Tobias Walter**[1], **Christian Wende**[2]

[1] ISWeb — Information Systems and Semantic Web
Institute for Computer Science, University of Koblenz-Landau
Email: {parreiras, walter}@uni-koblenz.de,
[2]Department of Computing Science, Technische Universität Dresden, Dresden
Email: c.wende@tu-dresden.de

17 August 2009

Abstract
Model Transformations are essential in order to support different kinds of models in a model driven environment. Model transformations are the main link between metamodelling technical spaces and ontology technical spaces. This deliverable describes frequently used patterns for transforming instantiations of metamodelling 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 in context of MOST.

Keyword List
software modelling, ontologies, transformation, composition, integration

Contents

Change Log

Version	Date	Author(s)	Changes
1.0	31.02.09	Fernando Silva Parreiras	Document created
1.1	30.06.09	Fernando Silva Parreiras	Document structure established
1.2	01.07.09	Christian	First version of Feature Model created
1.3	27.07.09	Christian	Documentation of Feature-based technology integration, Feature Mappings for integration features
1.4	31.07.09	Fernando Silva Parreiras, Tobias Walter, Christian Wende	Minor polishments
1.5	11.08.09	Fernando Silva Parreiras, Tobias Walter, Christian Wende	Addressed comments from internal review
1.6	17.08.09	Fernando Silva Parreiras	Addressed comments from internal review

List of Figures

List of Tables

1 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.

Reference Layer. Reference Layer is a set of abstract classes that are common for different packages. It defines the core elements of a given domain.

Implementation Layer. Implementation Layer is the set of classes that extend abstract classes in the Reference Layer by redefining or specifying their properties and operations.

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).

Mapping Rules. Mapping rules are relationships among constructs in two distinct metamodels.

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.

Scenario. Scenarios are outlines of set of use cases where the integrated metamodel happens to be used. They describe the users' work and are used to extract use cases from it.

1.1 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
NFA	nondeterministic finite automaton
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
RL	Reference Layer
TU	TwoUse
UML	Unified Modeling Language 2.0
WFR	Well Formedness Rules

2 Introduction

In software engineering, model-driven techniques (Model-Driven Engineering, MDE) have gained broad acceptance over the last decade. Model-driven techniques provide management of, transformations between and synchronisation of different models, including the “model” at the target platform constituted by source code. MDE is motivated by the objective of factorising complexity into different levels of abstraction and concern, from high-level conceptual models down to specific aspects of the target platform.

An instance of MDE is the Model Driven Architecture (MDA) [Miller and Mukerji, 2003], which is based on OMG’s Meta-Object Facility. It frequently includes UML as its modelling language and a common pipeline of managing and transforming models according to MDA [Kleppe et al., 2002] is depicted in the dashed box of Fig. 1: a platform-independent model (PIM) is transformed into a platform-specific model (PSM) and eventually into an executable representation (code) being the target platform. Thereby each transformation, i.e., each arrow, also indicates the enrichment of the resulting model with new features possibly specified in additional models.

Also over the last decade, the Web, AI and database communities have successfully investigated and promoted the use of *ontologies* as modelling and reasoning frameworks for the management of models and corresponding (Web) data. Ontologies and MDE technologies exhibit different foci. OMG MOF targets automating the management and interchange of metadata whereas knowledge representation focuses on semantics of the content and on automated reasoning over that content [Frankel et al., 2004].

In the past deliverables, we have proposed a combined metamodel based on requirements from the case studies and investigated the possibilities for querying the combined metamodel in order to access models in a flexible manner using existing languages. In this deliverable, we tackle the question of how one may define transformations from the visual languages into models conforming to the metamodels and eventually into target platforms and which transformation methods can be used repeatedly.

Like models, ontologies may provide a foundation for MDE. Thus, MDE can be based on the *Metamodelling Technical Space* (MMTS) as well as on *Ontological Technical Spaces* (OTSs). Kurtev et al. [Kurtev et al., 2002] have coined the term *technical spaces* to organise concepts in order to compare complex solution approaches. A technical space (TS) can be here understood as a body of knowledge comprising modelling languages and transformation facilities.

Subsequently, we will investigate the properties of ontological technical spaces, elucidating the potential of ontology technologies in MDE. Figure 1 illustrates an example indicating the use of several OTSs in the MDE process. The classical MDA transformations residing in the metamodelling technical space, such as explained above, are extended by further transformations making use of OTSs.

Further transformation into OTSs may provide additional analysis and implementation support, not as efficiently available in metamodelling technical spaces. Currently, MDA uses semi-formal metamodels instead of formal specification languages to describe models
[Tetlow et al., 2006]. In Fig. 1, the initial UML model representing the PIM is transformed into a MOST model (arrow 1). A MOST model describes the UML model together with OWL axioms, enabling logics-based model analysis. A second transformation translates the PIM into an OWL ontology and to Java Code (arrow 2 and 3). This model from OTS may serve as a kind of data base for a reasoner, invoked by the Java program.

In order to improve the understanding of the combined space composed by MMTS and OTS

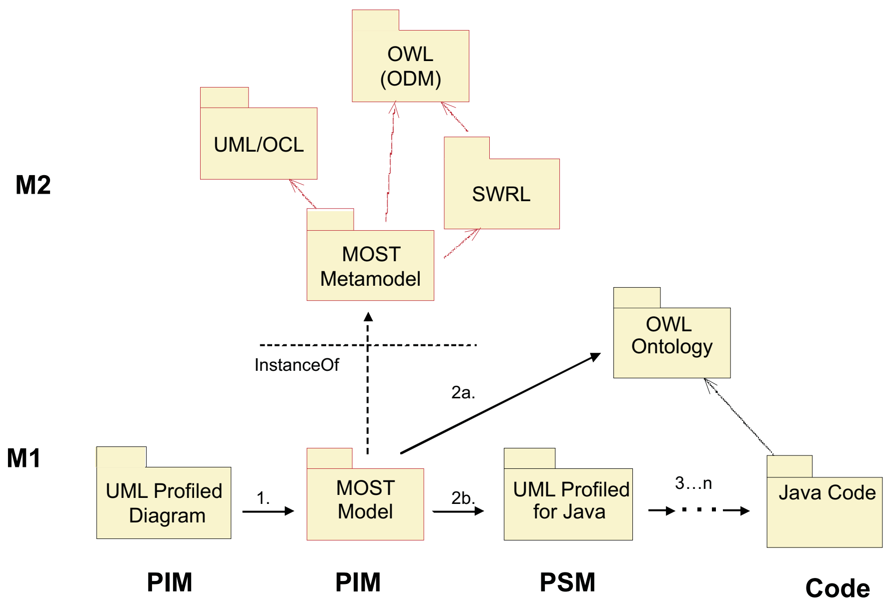


Figure 1: Marriage of MMTS and OTSs.

(MMTS+OTS), we compare different MMTS+OTS approaches appearing in MOST in relation to the MOST Workbench in Section 3. We define a feature model for technology integration in MOST in Section 5 that we derived from literature and MOST case studies cases. There, more details about arrows 1 to 3 will follow. Exemplary design patterns for technical integration are presented in Section 6.

3 MOST Workbench

The need for integration of MTTS and OTS that results from enabling MDE by ontology technology is an important factor in designing the MOST workbench. The workbench needs to provide means for connecting several metamodelling technologies and ontologies. Additionally, we aim at providing both a flexible and customisable architecture in order to support the specific requirements of the MOST Case studies (for details see [Friesen et al., 2009, Kasztelnik et al., 2009]) and further application scenarios.

3.1 Technology Integration in the Generic Architecture of the MOST Workbench

Integrating technology spaces is realised in specific layers of the generic Workbench Architecture [Bartho and Zivkovic, 2009]. Fig. 2 highlights these layers. As can be seen from the architecture, they are used by the upper layers of the workbench to access and manage modelling artefacts. Basically, integration of technology spaces means that the same artefact, e.g. a system model or an ontology, can be accessed from different technological spaces in order to apply a specific tool from an upper layer of the workbench, that was built for that technology

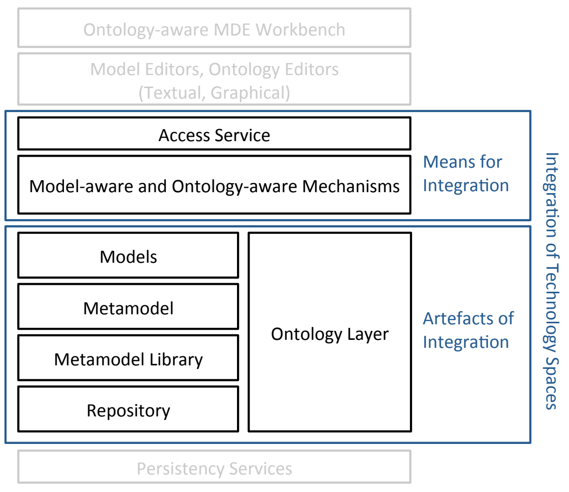

Ontology-aware MDE Workbench
Model Editors, Ontology Editors (Textual, Graphical)
Access Service
Model-aware and Ontology-aware Mechanisms
Means for Integration
Models
Metamodel
Metamodel Library
Repository
Ontology Layer
Artefacts of Integration
Integration of Technology Spaces
Persistency Services

Figure 2: Layers of the generic architecture of the MOST Workbench that are involved in integrating technology spaces.

space. Thus, we distinguish two general kinds of layers involved in integration:

Layers that provide means for integration These layers provide components for accessing artefacts of the technology spaces transparently though an access service. That means that the above layers can receive and use artefacts in the format of the technical space they were built for. In addition, the layers provide the technical infrastructure to perform transformations or adaptations of artefacts to exchange them between the technological spaces.

Layers that provide artefacts of integration These layers are divided into vertical units related to a specific technological space. Each vertical unit contains components to manage artefacts for a single technological space.

3.2 Customisation of the Generic Architecture for Technology Integration

The architecture shown in Fig. 2 is still rather generic. The layers and their responsibilities are not bound to a concrete metamodelling technology, an ontology technology, or a specific integration approach. Hence, the application of the MOST Workbench requires that the layers are refined and bound to concrete technologies. To come up with a tailored solution this binding should be determined by the requirements of the scenario the workbench is applied to. For example, the scenario determines the modelling languages or reasoning services that are needed. To systematise the tailoring of the generic MOST architecture for technology integration, to allow for component reuse in different scenarios, and exploit the extensibility of our generic architecture, it is reasonable to consider the MOST Workbench as a software product line of tools for ontology-aware MDE. A software product line (SPL) is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets.

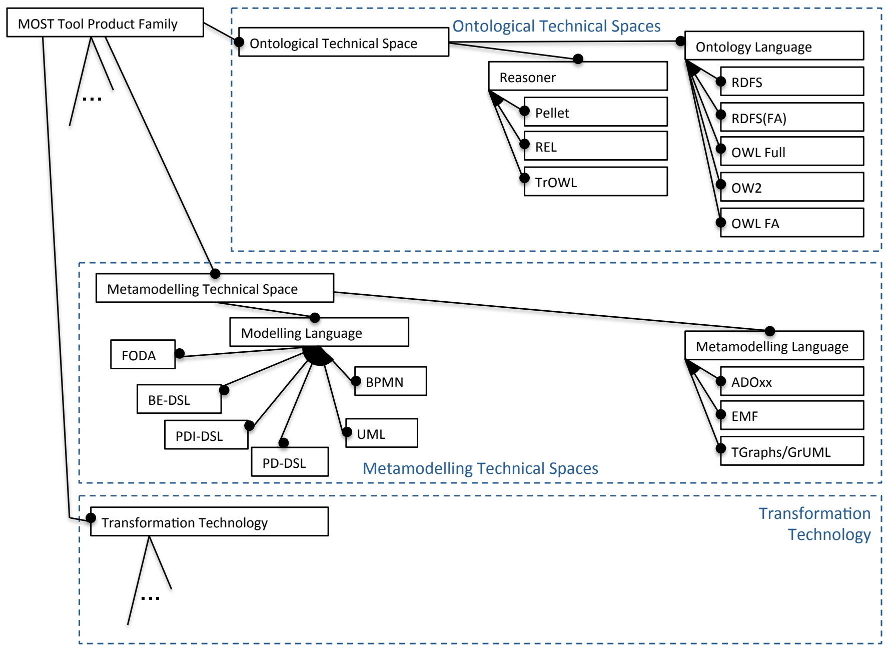


Figure 3: Features of the MOST Workbench that are related to the integration technology spaces presented as a feature model [Kang et al., 1990]. From a group of features connected by a filled arc at least one feature needs to be selected. A non-grouped feature connected by a line ending with a filled circle is mandatory.

In addition to the shared core assets, every system of a SPL has features that are specific to the system and that are not shared by all other systems (often called products) of the SPL. To express this variability, feature modelling can be used to describe the different features available in an SPL and their interdependencies.

An application of product-line engineering for the MOST Workbench requires a profound analysis and specification of the commonalities and variability of the workbench product-line. Fig. 3 depicts a fragment of the feature model we derived from the findings presented in [Bartho and Zivkovic, 2009] for analysing the technological variation space in MOST. In this document we focus on features that are related to the the integration of technical layers. More information on the details of applying feature-based customisation to build the MOST Tool Product Family (MOST TOPF) can be found in [Srdjan Zivkovic, 2009]. In the feature model we indicated different regions that are related to a specific dimension of variation for technical integration.

Variability modelling resides in the *problem space* whereas the realisation of features is part of the *solution space*. To instantiate products from our SPL, feature realisations in the solution space have to be included according to the presence of the features in a variant model. Thus, we have to fill the generic technical integration layers of the generic architecture with concrete technology components and have to specify how these components relate to features of our feature model. This mapping enables an automatic instantiation of the technical integration infrastructure for products from the MOST TOPF. For example, the selection of a special modelling language would determine the inclusion of the language components and the corresponding metamodelling infrastructure. The realisation of this mapping requires a deep understanding of the technical spaces. Since variation within the metamodelling and ontological technical spaces was already analysed and discussed in previous MOST deliverables, both regions can be derived quite straightforward. A gap in the feature model for technological integration that will be closed by this deliverable is variability on the used transformation technology.

The rest of this document we will investigate and refine the regions identified in our feature model and discuss their characteristics. First, Sect. 4 provides a detailed discussion of the regions of our feature model that relate to technical spaces already identified in the context of MOST. It reports on the metamodelling infrastructures used in MOST in Section 4.1 and on the variability found for ontology technology in 4.2.

In Sect. 5 we contribute new features that are required to configure transformations between technological spaces by analysing the variability found in the technology transformations already used in the context of MOST. Based on this variability analysis we extend the feature model in Figure 3 and derive some design patterns for transformation approaches in Sect. 6.

Sect. 7 shows how such a model-based approach of mapping features to models can be exploited to customise the technological integration in the MOST TOPF for specific application scenarios. We map features to components of the workbench and describe how this enables the derivation of customised technical integration layers for a concrete product of the MOST TOPF.

4 Technical Spaces in MOST

In this section we identify and analyse which technical spaces are considered in the MOST project. Here we distinguish between metamodelling and ontological technical spaces. In general we assume the following definition of Technical Space:

A technical space is a model management framework accompanied by a set of tools that operate on the models definable within the framework.

This definition leads to a commonality of technical spaces. Each technical space contains a set of models, organised in a model management framework. This framework defines the conceptual foundations and notations for creating models within the technical space.

Further relevant is the structure of a technical space. In different work it appeared that the models in the technical space can be connected by certain relations and form a layered architecture [Bezivin and Kurtev, 2005]. The technical space usually is based on one single model that is used to create other models in the space. Models can be related via *conformsTo* relations that distribute the models across different layers in the architecture. In the following we consider a layer-architecture with three different layers, namely the M1-, M2- and M3 layer. The model at the *M3 layer* is called *metametamodels* and conforms to itself. The ones at the *M2 layer* are the *metamodels* and conform to the metametamodel. The *M1 layer* contains the *models* which conform to the metamodels at the M2-layer.

Models at every level are related to a model at the upper level via the *conformsTo* relation. This three-layer organisation is the foundation for a technical space managing different models. It mainly bases on a fixed metametamodel at the M3 layer and the meaning of the *conformsTo* relation between the layers. Thus, in the following for each technical space in the MOST project we will give the idea of the metametamodel.

4.1 Metamodelling Technical Spaces

In this section we will consider metamodelling technical spaces which are available in the MOST tool product family. In the following we describe the *Meta Object Facility* (MOF), the *ADOxx technical space* and *GrUML*, a space for modeling with graphs.

4.1.1 MOF

Whereas *models* describe a specific abstraction of reality (cf. e. g. [Apostel, 1960]) *metamodels* define the modelling itself including applied modelling technologies and modelling processes [Brinkkemper, 1996]. Today, the metamodelling space is associated with technologies developed in the UML environment [OMG, 2009b]. Here, metamodelling is usually restricted to specify modelling (and programming) languages only.

Metamodel-based approaches are based on a staged architecture of models and metamodels, where the structure of lower level models is explained by higher level metamodels. The Meta Object Facility [OMG, 2000] defines a four-layer structure, which is applied to define domain specific languages and general purpose languages like UML. At the M3 layer MOF is defined by a itself, since MOF is itself metametamodel. Language specifications like the UML specification are viewed as (linguistic) instances [Atkinson and Kühne, 2003] of MOF and reside on the metamodel level (M2). The model level (M1) contains concrete models, like class-diagrams or state machines, the notation of which is defined by metamodels on M2. Finally, M0 contains instances representing real world objects.

The metametamodel provided by OMG technical space is called *Meta Object Facility* (MOF). It provides a language for defining the abstract syntax of modelling languages. MOF is in general a minimal set of concepts which can be used for defining other modelling languages represented by M2 metamodels. 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. We see that EMOF mainly consists of the UML Basic package, whereas CMOF extends EMOF by using the Constructs package (both part of the UML infrastructure [OMG, 2003]).

An implementation of the MOF technical space is the Eclipse Modeling Framework [Budinsky et al., 2003]. EMF is an open-source implementation with code generation facility. Although we present a conceptual description of MOF, in MOST we consider EMF in the implementation.

Thus, EMOF and Ecore can be consider de facto equivalent. In the next sections we use Ecore as an practical implementation of EMOF.

Models in the metamodelling technical spaces are viewed as object networks or graphs. Their abstract syntax is usually defined by (UML-)class diagrams extended by constraint languages like OCL [OMG, 2005]. Furthermore, UML provides profiles to extend the set of UML modelling primitives for specific domain needs. Profiles are based on stereotypes, which extend the concepts specified by an appropriate UML metamodel [OMG, 2009b]. UML profiles are used e. g. to specify the Ontology Definition Metamodel (ODM) [OMG, 2009a], a family of UML metamodels defining various ontology languages including OWL.

In the following we want to consider some metamodels which conform to MOF:

- *UML:* The Unified Modeling Language provides a set of general-purpose modelling languages. The provide notations for modelling of structures, behaviors and interactions. All languages mainly conform to MOF.
- *KM3:* KM3 (KernelMetaMetaModel) is a domain-specific language. It is used to describe EMOF based metamodels in a textual manner. KM3 conforms to itself, thus provides a technical space.
- *BPMN:* The Business Process Modeling Notation (BPMN) is a graphical representation for specifying business processes in a workflow. The language itself conforms to EMOF and thus also can be described by KM3.
- *BEDSL:* The Business Entites DSL (BEDSL) is a domain-specific language intended to model business entities. It provides concepts like Entity and Attribute and conforms to itself.
- *PDDSL:* Physical Devices Domain Specific Language (PDDSL) enables the specification of possible configurations of the device models. The language provides concepts related to the structure of a device: Configuration, Slot, Card. The language conforms to EMOF.
- *PDIDSL:* Physical Device Instances Domain Specific Language (PDIDSL) enables definitions of concrete instances of devices that conform to the PDDSL specifications.
- *FODA:* FODA (Feature-oriented domain analysis) provides notations for building feature models. Such models define features and their dependencies, typically in the form of a feature diagram. The language conforms to EMOF.

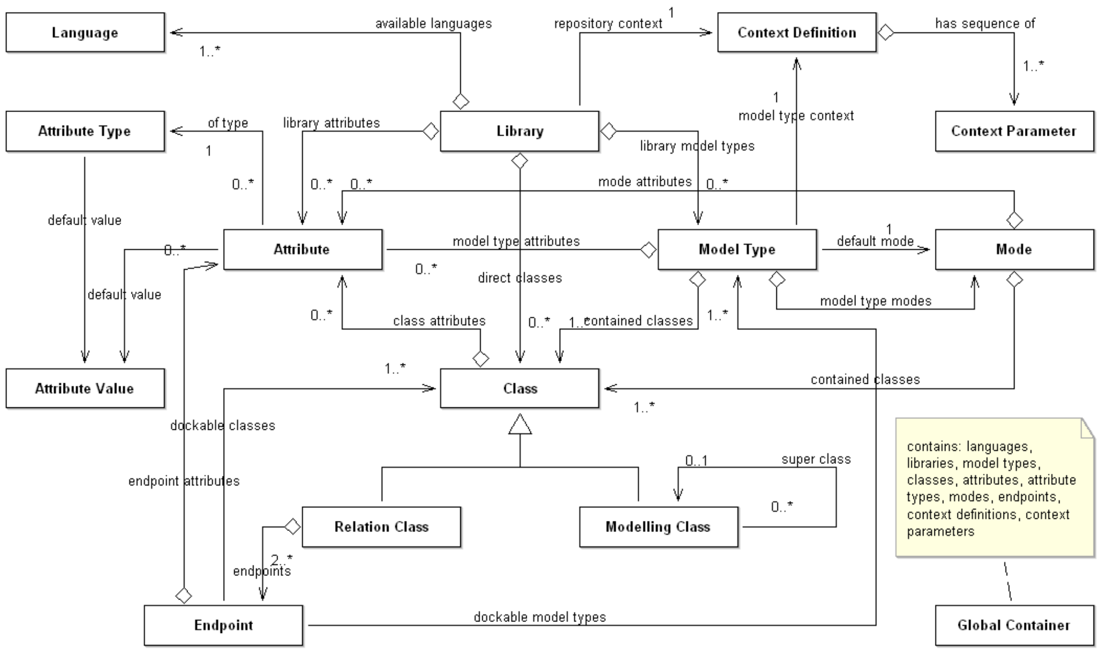


Figure 4: ADOxx Meta^2 model

4.1.2 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.

Figure 4 depicts an excerpt of the *ADOxx Meta^2-Model*. Here a *class* represents the description of a particular modelling object. Classes are connected by *relation classes* via *endpoints*. An endpoint defines which classes, relation classes or model types the relation class can connect to.

An *attribute* describes a property metamodel elements may have. Each attribute has to be of some *type* and has to have a default *value*. A *model type* is a container of classes and relation classes. It can be compared to a diagram concept in UML, but for the domain specific languages it might represent the metamodel concept itself. A library is a container of model types and transitively of all metamodel elements needed for a product metamodel.

A detailed definition of the ADOxx Meta2-Model can be found in [Bartho and Zivkovic, 2009].

4.1.3 GrUML

Graphs are well-defined mathematical and formal structures. Different algorithms and methods exist to work efficiently on graphs. Normally, graphs appear all-around in today's software engineering. For example and for simple imagination, UML class diagrams also can be seen as a graph, where the nodes are represented by classes and the edges by associations. A very

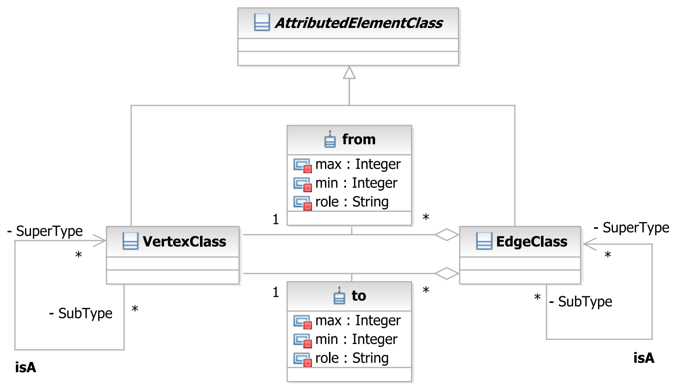


Figure 5: Part of the GrUML Metametamodel

general kind of graphs are the TGraphs [Ebert et al., 2002]. Such graphs are *directed*, its edges and vertices are *typed* and *attributed* and for each node the incident edges are *ordered*. Each graph is an instance of its corresponding *schema* (metamodel) which for example defines types of edges and vertices and structures them in hierarchies.

To define graph schemas a metametamodel called *GrUML* (Graph Unified Modelling Language) is used. Sets of vertices of the graph are represented by classes with attributes. Sets of edges are represented by associations which can contain attributes too.

A GrUML diagram is a visualisation of a TGraph model. A TGraph model itself is instance of the GrUML-metametamodel at the corresponding M3 layer. Figure 5 depicts an excerpt of the GrUML metametamodel which lies at the M3-layer of the three-layer architecture. Here we have one class for vertexes and one class for edges. Each edge connects two vertex classes. `EdgeClass` and `VertexClass` are specialisations of `AttributedElementClass` which means, that both can be attributed.

Using the GrUML metametamodel we can create instances of it at the M2-layer. TGraphs are composed of such instances whose type for example is `VertexClass` or `EdgeClass`.

For detailed information see [Bildhauer et al., 2008, Bildhauer et al., 2007].

4.2 Ontological Technical Spaces

In this section we will consider ontology technical spaces which are available in the MOST tool product family.

An ontology constitutes a formal conceptual model. Hence, its core concerns, i.e. formal definitions of classes and relationships, are germane to the software engineering community. Given their roots in knowledge representation and reasoning, however, ontologies have always been used differently than conceptual models in software and data engineering. Hence, the perspectives on modelling and using ontologies are slightly twisted if compared to conceptual models such as UML class diagrams.

4.2.1 Ontology modelling

The process of modelling ontologies exhibits a couple of overlaps with the development of conceptual models [Staab et al., 2001]. Requirements elicitation is followed by a design phase when classes and relationships need to be defined similarly as in an UML class diagram. This stage, however, is followed by another step that depends on the ontology modelling paradigm and its corresponding language.

In the realm of description logics-based ontologies [Baader et al., 2003], the strength of ontology modelling lies in disentangling conceptual hierarchies with an abundance of relationships of multiple generalisation of classes (cf. [Rector et al., 2004]). For this purpose, description logics allows for *deriving* concept hierarchies from logically precisely defined class axioms stating necessary *and* sufficient conditions of class membership.

In the realm of logic programming-based ontologies [Angele and Lausen, 2004], the strength of ontology modelling lies in a formally integrated consideration of expressive class and rule definitions.

In both paradigms, the structure of class definitions may be validated by *introspecting* the model using corresponding reasoning technology. In the first model of description logics this is indeed the focus of its reasoning technology, while in the second model the focus of the corresponding reasoning technology is on reasoning with objects in a logical framework.

4.2.2 Ontology Languages and Reasoning

The language and reasoning paradigm that has been predominantly used and researched is the family of description logics languages, including the W3C recommendation, the Web Ontology Language (OWL) [Mcguinness and van Harmelen, 2004]. All description logic languages allow for capturing the schema in the "terminological box" (T-Box) and the objects and their relationships in the "assertional box" (A-Box).

The individual members of this family of languages differ in the set of modelling constructs they support. Depending on the exact configuration of allowed modelling primitives a member of the family like DL Lite [Calvanese et al., 2005], the W3C recommendations OWL-lite and OWL-DL (Web Ontology Language), and KL-One belongs to the class of languages requiring polynomial, Exptime, Nexptime and undecidable sound and complete reasoning algorithms, respectively.

Ontology languages that are derived from logic programming are usually Turing-complete [Angele and Lausen, 2004], but with their focus on instance reasoning act and work rather like deductive databases. Recent research investigates the integration of the two paradigms of description logics and logic programming [Motik and Rosati, 2007].

4.2.3 RDF and RDFS

The Resource Description Framework (RDF) [Klyne and Carroll, 2004] was developed for the description and annotation of web resources. The main target of this language was to exploit the full potential of the web (cf. [Berners-Lee, 1997, Berners-Lee, 1998]). Within this process, one step is to add semantics to the information in order to provide a meaningful, machine-understandable knowledge representation.

RDF Schema (RDFS) can be seen as a first try to support expressing simple ontologies with RDF syntax. In RDFS, predefined Web resources `rdfs:Class`, `rdfs:Resource`, and `rdf:Property` can be used to declare classes, resources, and properties, respectively. RDFS predefines the following metaproperties that can be used to represent background assumptions in ontologies: `rdf:Type`, `rdfs:SubClass`, `rdfs:SubPropertyOf`, `rdfs:domain`, and `rdfs:range`. At a glance, RDFS is a simple ontology language that supports only class and property hierarchies, as well as domain and range constraints for properties.

4.2.4 RDFS(FA)

RDFS(FA) [Pan and Horrocks, 2007] is a sub-language of RDF(S), which provides a Fixed layer metamodelling Architecture for RDFS. RDFS(FA) provides a UML-like metamodelling architecture. Let us recall that RDFS has a nonlayered metamodelling architecture; resources in RDFS can be classes, objects, and properties at the same time, namely, classes and their instances (as well as relationships between the instances) are the same layer. RDFS(FA), instead, divides up the universe of discourse into a series of strata (or layers). The built-in modelling primitives of RDFS are separated into different strata of RDFS(FA), and the semantics of modelling primitives depend on the stratum they belong to. Theoretically, there can be a large number of strata in the metamodelling architecture; in practice, four strata are usually enough. The UML like metamodelling architecture makes it easier for users who are familiar with UML to understand and use RDFS(FA).

4.2.5 OWL Full

OWL Full [Smith et al., 2004] is the most expressive language of the OWL languages. It is syntactically and semantically upward compatible with RDF(S) and the other OWL languages. The syntax of OWL Full allows the combination of all OWL, RDF and RDFS language primitives. The disadvantage of this expressive language is the high worst case complexity. There is no reasoner which is able to handle the full power of the language and OWL Full is undecidable.

4.2.6 OWL FA

OWL FA [Pan et al., 2005] enables metamodelling. It is an extension of OWL DL, which refers to the description logic $\mathcal{SHOIN}(\mathcal{D})$. Ontologies in OWL FA are represented in a layered architecture. This architecture is mainly based on the architecture of RDFS(FA) [Pan and Horrocks, 2003]. OWL FA specifies a stratum number in class constructors and axioms to indicate the strata they belong to.

4.2.7 OWL2

OWL2 [Motik et al., 2008] provides simple metamodelling which correspond to the contextual semantics defined in [Motik, 2007]. However, this modelling technique is mainly based on punning. It has been shown in [Pan et al., 2005] that this can lead to non-intuitive results, since the interpretation function is different based on the context.

5 Feature Model of Transformation Technology in MOST

A transformation definition is a set of transformation rules that together describe the conversion of one model in the source language into another related model in the target language [Kleppe et al., 2002]. Concerning MMTS+OTS, we distinguish between different aspects of transformations as follows.

5.1 Identification of Features in the Transformation Technology Space

In the following section, we describe features present on transformations between MMTS and OTS. Figure 6 presents the resulting feature model.

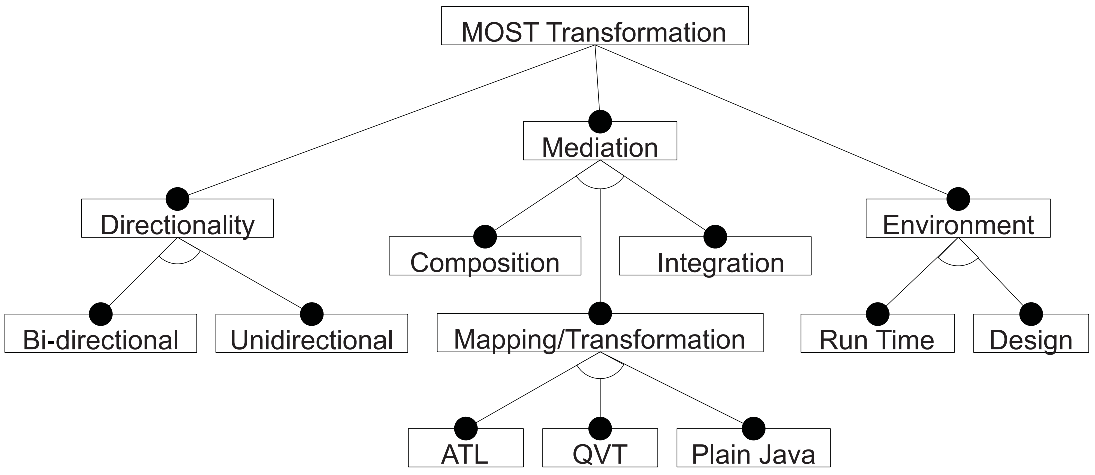


Figure 6: MOST Transformation Feature Model

5.1.1 Mediation

Mediation is the process of reconciling differences between heterogeneous models. It plays a central role in MMTS+OTS, as models in different languages must coexist. Features of mediation are:

1. *Mapping/Transformation*: the declarative specification of the correspondences between different elements of the two models. In a transformation process, the mapping specification precedes the transformation definition.

2. *Integration*: focuses on interoperability between models so that they work together effectively. It comprises:

    - Aligning: preserves the source models and produces a new model containing additional axioms to describe the relationship between the concepts from the source models.
    - Merging: creates a new merged model based on concepts found in the merged source models.

3. *Composition*: comprises the combination of elements that accord to overlapping concepts in different source models. Usually, each source model handles a different dimension of the overlapping elements. A weaving process does not necessarily produce a merge, but it can produce a model in a third language with new knowledge based on the source models.

Both integration and composition make use of mappings to specify overlaps.

For example, the alignment of a UML class diagram and an ontology could involve the specification of the mappings between both models. The resulting model can be queried transparently for classes from both models. Merging involves the generation of a completely new model. Both integration strategies consider relating only the similar concepts of each language.

A composition can generate a model in a new language with the expressiveness of the previous languages by unifying the concepts of both languages. For instance, in models defined in a language that is composed from UML and ontologies this would allow for specifcing UML methods in an ontological class definition.

5.1.2 Environment

The composition of the TSs can occur either at runtime or at design time. The usage of ontologies at development time that describe the problem domain itself is classified by [Happel and Seedorf, 2006] as Ontology-driven development (ODD), whereas using an ontology as a primary artefact at runtime is classified as Ontology-based architectures (OBA).

Please clarify these points ODD brings knowledge-based foundation to enable reasoning. Other possible applications include validation and automated consistency checking.

Reasoning is also an added value in OBA approaches. When applying ontologies as Domain Object Model, reasoning services can be used...

5.1.3 Directionality

Directionality concerns the ability to transform models in different directions based on a single transformation definition and the notion of source and target language.

Unidirectional transformation allow for transformation execution in only one direction by creating or updating a target model from a given source model. An approach that falls in this category is ATL.

Bidirectional transformations enable forward and backward transformations between source and target models. They are especially useful for synchronisation of models. Examples for bidirectional transformation approaches are QVT and UMLX.

6 Patterns for Model Transformations

In this section, we describe transformations patterns between metamodelling and ontology-based approaches. We have formed five groups of approaches and within each group we delve into some details of a "prototypical" approach, which is also classified according to the framework defined in Section 5: transformation of languages with similar constructs, transformation of langugages with different constructs, transformation for model checking, transformation for model enrichment and transformation for extending model expressiveness. The reader may note that the variation is indeed larger than can be fully covered by this description.

6.1 Transformation of languages with similar constructs

When a metamodel has constructs similar to the OWL metamodel, a transformation of these constructs can be used for semantic validation of the source model or extending its functions.

Ecore2OWL The Ecore2OWL transformation is responsible for mapping Ecore-based metamodels to an ontology, in order to formalise their constraints and semantics. Therefore, the metamodel is transformed to an ontologies' TBox. Model instances are transformed to the corresponding ABox and the ontology is checked for consistency.

Similar transformations that fit into this category are KM32OWL, Meta2model2OWL, BEDSL2OWL. KM32OWL, Meta2model2OWL and BEDSL2OWL correspond to transformations of an metamodel specified using KM3, Meta2mode, and BEDSL respectively into an OWL ontology.

- Transformation Technology: ATL or Java
- Metamodelling Language: ADOxx, EMF
- Environment: Design Time
- Ontology Language: OWL2
- Modelling Level: M2
- Mediation: Transformation
- Directionality: Unidirectional
- Metamodelling Language: Ecore
- Ontology Language: OWL2

Another example of transformation that fits in this category is TGraph2OWL. The goal of the transformation TGraph2OWL is to map TGraphs to ontologies for different reasoning tasks. The schema of a graph is transformed into an ontologies' TBox. The corresponding ABox results from the graph itself which is an instance of the schema.

6.2 Transformations of languages with different constructs

BPMN to OWL : To get a formal description of the control flow of a BPMN process or to query processes with specified control flow characteristics and modalities, BPMN diagrams are transformed to ontologies. The ontologies' T-Box represents the BPMN diagram, while the corresponding A-Box consists of possible executions of the process (execution set semantics).

In the following, we describe a concrete configuration of the transformation corresponding to the feature model identified in section 5.

- Transformation Technology: Java
- Metamodelling Language: Ecore
- Ontology Language: OWL2
- Modelling Language: BPMN
- Environment: Design Time
- Modelling Level: M1
- Mediation: Transformation
- Directionality: Unidirectional
- Metamodelling Language: Ecore, ADOxx
- Ontology Language: OWL2

Feature Model to OWL In feature modelling, a variant model derived from a feature model represents a concrete instance of a product (i.e., a specific system in a domain). The approach presented in [Wang et al., 2007] uses OWL classes to simulate features. A transformation is used to map features and their relationships to an OWL DL TBox for verifying a variant against its feature model.

In the following we describe a concrete configuration of the transformation corresponding to the feature model identified in section 5.

- Transformation Technology: Java
- Modelling Language: FODA, OWL2
- Environment: Design Time
- Modelling Level: M1
- Mediation: Transformation
- Directionality: Unidirectional
- Metamodelling Language: Ecore
- Ontology Language: OWL2

PDDSL to OWL The goal of the PDDSL2OWL transformation is to extract the OWL T-box from the PDDSL models. The transformation maps concepts of the PDDSL to OWL classes and properties. Furthermore, it specifies a formal semantics of the concepts of PDDSL (e.g. formalisation of configuration constraints).

In the following we depict a concrete configuration of the transformation corresponding to the feature model identified in section 5.

- Transformation Technology: QVT
- Modelling Language: PDDSL
- Environment: Design Time
- Modelling Level: M2
- Mediation: Transformation
- Directionality: Unidirectional
- Metamodelling Language: Ecore, ADOxx
- Ontology Language: OWL2

PDIDSL to OWL The PDIDSL2OWL transformation takes as input a model containing the specification of physical devices expressed in PDIDSL. It is transformed to the respective OWL A-box. The transformation updates the ontology T-box produced by the PDDSL2OWL transformation (cf. above paragraph) by adding individuals along with the respective facts assertions.

In the following we depict a concrete configuration of the transformation corresponding to the feature model identified in section 5.

- Transformation Technology: QVT
- Modelling Language: PDIDSL, OWL2
- Environment: Runtime
- Modelling Level: M1
- Mediation: Transformation
- Directionality: Unidirectional
- Metamodelling Language: PDDSL, ADOxx, Ecore
- Ontology Language: OWL2

6.3 Transformation for Model Checking

This category groups the works that use automated reasoning techniques for checking and validation of models in formalised languages. Most reasoning approaches for validation check some specification against some design. The description logics technical spaces, however, have specifically been defined to validate the *internal* consistency of a set of class definitions. To exploit this form of validation, one may transform a part of a given MDE-based model, e.g., an UML class diagram, into a set of OWL class definitions (cf. arrow 1 and 2a in Fig 1; cf. [Berardi et al., 2005]) and one then check class hierarchy, the property hierarchy as well as the logical consistency of instantiating classes.

We illustrate this process by using the simple UML class diagram of university accounts depicted in Fig. 7. The diagram shows that a `WebPortalAccount` is a particular kind of `UserAccount` and that each `UserAccount` is owned by one and only one `User`. Additionally, a `User` can be only of two different kinds, a `Researcher` or a `Student`. A `Researcher` can have only one `WebPortalAccount`. The association class `Uses` specialises the association class `Owns`.

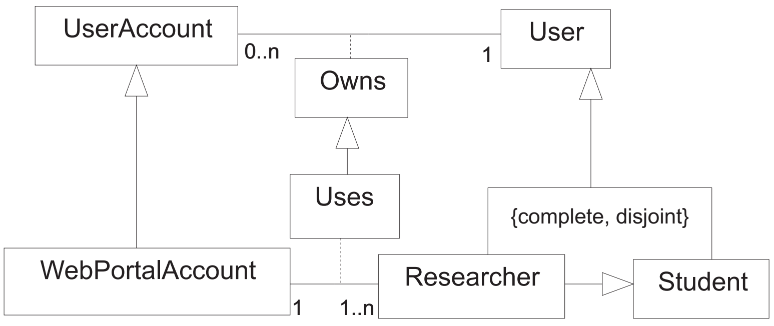


Figure 7: Checking consistency of UML models.

After applying the transformation from UML into a description logics model, such as OWL (more specifically, [Berardi et al., 2005] mapped it into $\mathcal{ALCQI}$), we use the reasoner to verify the model. Reasoning over the model discovers some inconsistent properties. First of all, the class `Researcher` must be empty and, hence, cannot be instantiated. The reason is that the disjointness constraint asserts that there is no `Researcher` who is also `Student`. Furthermore, since the class `User` is made up by the union of classes `Researcher` and `Student`, and since `Researcher` is empty, the classes `User` and `Student` are equivalent, implying redundancy.

Dropping the generalisation `Student-Researcher` results in a valid model. If we invoke the reasoner again, we can refine the multiplicity of the role `Researcher` in the association `uses` to 1. `Owns` is a generalisation of `Uses`, hence every link of `Uses` is a link of `Owns`, since every `Account` is owned by exactly one `User`, necessarily every `WebPortalAccount` is used by at most one `Researcher`, since `WebPortalAccount` is a subclass of `Account`.

Reconsidering our feature model depicted in Fig. 6, the configuration of this category uses the following features: (i) a model at M2 or M3 modelling level; (ii) a target model, written in an ontology, reasoning capability; (iii) a mapping specification describing the links between the models; (iv) a unidirectional transformation definition.

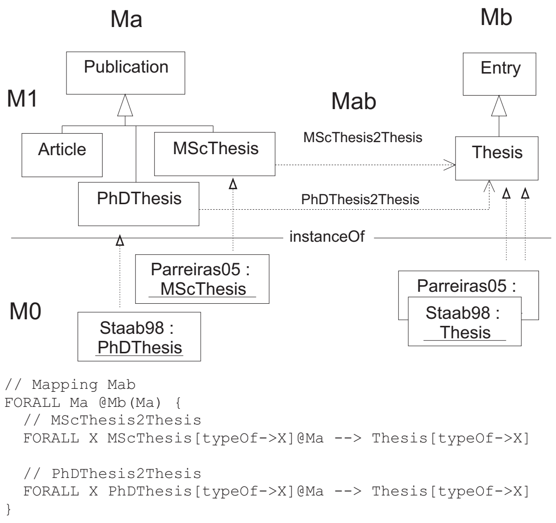
Another work, that fits into this category of MMTS to OTS transformation for the purpose of model checking, is [Straeten et al., 2003]. It proposes an approach to detect and resolve inconsistencies between different versions of UML models specified as a collection of class diagrams, sequence diagrams and state diagrams. It presents a Domain Specific Language (UML Profile) able to describe the evolution of the models.

6.4 Transformation for Model Enrichment

Model enrichment aims to use ontologies for enrichment of models from MMTS. This category encompasses the approaches that make use of ontologies to infer knowledge from the MMTS models and convert these inferences back as facts in the new MMTS models. The main difference between this category and the former is the bidirectional transformation and the application of logical rules and reasoning on the OTS side. First, the MMTS model is transformed into a OTS model. On the OTS side, inference services take the transformed model and the logical rules to make explicit the knowledge not present in the MMTS. Then, the resulting OTS model is transformed back.

One example to illustrate this category is the usage of the TRIPLE framework. TRIPLE [Decker et al., 2005] is a reasoner for which a hybrid rule language with lexical notation for querying and translating RDF models has been designed. It is primarily based on logic programming and has strong ties with F-Logic. By reasoning on the TRIPLE model one may derive new facts to be transformed back to and included in MMTS, e.g. in the PIM.

Figure 8 illustrates a simplified example: (Step 0: Model) It depicts two models capturing bibliographical references. On the left side, the model `Ma` comprises the class `Publication`, which generalises `Article` and `Thesis`, which generalises `MScThesis` and `PhDThesis`. On the right side, the model `Mb` includes the classes `Entry` and `Thesis`. In the middle, there is a mapping model, `Mab`, with a link `MScThesis2Thesis` mapping `MScThesis` onto `Thesis` and a link `PhDThesis2Thesis` mapping a `PhDThesis` onto a `Thesis`.



```
// Mapping Mab
FORALL Ma @Mb(Ma) {
  // MScThesis2Thesis
  FORALL X MScThesis[typeOf->X]@Ma --> Thesis[typeOf->X]

  // PhDThesis2Thesis
  FORALL X PhDThesis[typeOf->X]@Ma --> Thesis[typeOf->X]
}
```

Figure 8: Mapping between two models Ma and Mb.

(Step 1: Forward Transformation) The three models are transformed into the TRIPLE language from the OTS. The resulting TRIPLE model includes all instances and classes of `Ma` and `Mb` as RDF and it contains the two logical rules depicted in the lower part of Fig. 8.

(Step 2: Logical Querying) The logical rules allow for querying of `Ma` instances through the view of `Mb` in the TRIPLE OTS [Decker et al., 2005]. The corresponding query is defined in TRIPLE by `FORALL X,Y,Z <- X[Y->Z]@Mb(Ma).` The retrieved instance triples may be added as triples to the RDF space of the `Mb` part of the TRIPLE OTS.

(Step 3: Backward Transformation) Eventually, one may transform the latter results back to `Mb`, then including all the objects of `Ma` as seen through `Mb` in `Mb`.

While the given example is too simple to be of use in the software engineering process, real applications may exploit the TRIPLE inferencing and enrichment *(i)* to translate (database) objects between PSM/code models at run time, or *(ii)* to perform more complex reflections (i.e. at the model level) that need the help of logic programming, e.g. recursive logical rules such as exploited in [Oberle et al., 2004].

Regarding the feature model, the configuration of features include: (i) a model at M2 or M3 modelling level; (ii) a target model, written in OWL2, reasoning capability; (iii) a mapping specification describing the links between the models; (iv) a *bidirectional* declarative transformation definition; (v) logical rules and reasoning to make the knowledge explicit on the OTS side.

As this is not a closely grouped category, further work in this category exhibits strongly varying facets. Billig et al. [Billig et al., 2004] use TRIPLE to generate mappings between a PIM and a PSM describing user requirements as input. It comprises a transformation from MMTS into OTS (TRIPLE), the generation of the mappings, the transformation into a PSM under OTS and the transformation OTS to MMTS of the PSM. Roser and Bauer [Roser and Bauer, 2006] present a framework to automatically generate model transformations between MMTS models using the OTS; Kappel et al.[Kappel et al., 2006] provide an approach for model-based tool integration. It consists of transforming two metamodels from MMTS into an OTS, uses reasoning services and generates mappings between the two models represented in the OTS.

6.5 Transformation for Extending Model Expressiveness

This category embraces our attempt at considering behavioural and representational aspects of modelling an application at MMTS (arrows 2a 2b in Fig. 1), called TwoUse (Transforming and Weaving Ontologies and UML in Software Engineering) [Silva Parreiras et al.,]. It involves: (i) a model written in profiled UML with OCL expressions; (ii) a target model, written in OWL, and; (iii) a *composition* including a mapping specification describing the links between the models; (iv) a bidirectional transformation definition. It differs from the former category as there is not one target model, but rather the aim is to eventually have models for code *as well as* for a logical space queried during run time.

Using a UML Profile for ontology, modelers design only an ontology, but cannot design an object-oriented model in *the same diagram.* With TwoUse, modellers use OCL-like expressions to describe query operations in the same diagram. Moreover, these operations can query the ontology, i.e., invoke a reasoning service at run time that uses the same ontology.

The ontology can be directly generated from the model (PIM) (arrow 2 in Fig. 1), whereas the object-oriented classes and OCL expressions are translated into a specific platform (arrow 2b) and later into Java code (arrow 3) including the API for ontology and reasoning invocation.

	Validation	Enrichment	Extended Expressiveness
Similar Constructs	UML2OWL, Ecore2OWL, Meta2Model2OWL	BEDSL2OWL, Ecore2OWL	TwoUse, OntoDSL
Different Constructs	BPMN2OWL, Feature2OWL	Languages modeled by OntoDSL	-

Table 1: Relation Transformation Patterns and Variants.

6.6 Combining MOST Transformation Patterns

In this section we relate the transformation variants identified in Section 6.1 and 6.2 to the discussed transformation patterns.

Table 1 relates the variants with the model transformation design patterns. Some transformations like Ecore2OWL can be seen in both patterns Validation and Enrichment. Indeed, it is possible to use ontology technologies for only validating models or to infer information from the models. In the latter case, the model transformation specification is responsible for translating the information inferred back to the Ecore model.

New approaches developed in MOST can be classified as approaches with extended expressiveness and approaches involving languages with different constructs. Thus, MOST contributes directly for extending the realm of patterns to new categories.

7 Realisation of Technological Integration by Components of the MOST TOPF

In this Section we will discuss how the previously identified features for technological integration are realised within the MOST Workbench. We will use the model-based approach for mapping features to components of the MOST Tool Product Family (MOST TOPF) introduced in [Srdjan Zivkovic, 2009]. As described in [Srdjan Zivkovic, 2009] this mapping enables the feature-driven customisation of a specific MOST product from the MOST TOPF.

The component diagram in Fig 9 illustrates that the realisation of technological integration becomes apparent in the `Access Layer` and the layer of `Model-aware and Ontology-aware Mechanisms`. The `Access Service` contributes the `Model Provider` component which provides transparent access to concrete models in the format and language that components in the above layers require. It also hides the technological integration needed to transform models between different technological spaces if necessary. For that purpose, the `Model Provider` uses the services provided by the `Transformation Broker` also located in the `Access Layer`. The `Transformation Broker` manages and coordinates all model transformation components contributed by the layer of `Model-aware and Ontology-aware Mechanisms`. If a model is requested in a specific format or language which it is not yet available in, these transformation components realise the needed technological transformation.

To specify the the mapping of technology integration features in the feature model to realisation components of our architecture we used the tool FeatureMapper [Heidenreich et al., 2008b, Heidenreich et al., 2008a]. It allows for defining mappings of features to model elements defined in arbitrary Ecore-based languages. The FeatureMapper provides an interactive mapping process and stores the mappings in a dedicated model. The mapping model is used to support the developer of the product-line by different visualisation techniques and for an automatic product derivation from a variant model. More on the overall process of feature-driven variant derivation can be found in [Srdjan Zivkovic, 2009].

Figure 10 shows how the FeatureMapper is used for mapping features of the MOST TOPF to MOST workbench components. The Assigned Feature Expression in Compartment (4) tells us that we have already defined a mapping between the feature *Ecore* and the components `Ecore` and `EMF` that are selected in the UML component diagram (5). This mapping means, that whenever the feature *Ecore* is selected for a product variant these components have to be included in the workbench instance for this product.

7.1 Mapping Features of the Metamodelling Technical Space

In the layer of `Model-aware and Ontology-aware Mechanisms` the inclusion of a (transformation) component is typically driven by the inclusion of a specific modelling or metamodelling language in a MOST TOPF product. Furthermore, the selection of a language triggers the inclusion of the metamodelling technology it is based on. Thus, we used the FeatureMapper to map every feature representing a modelling or metamodelling language in the feature model to the components it requires. We used the Context View visualisation of the FeatureMapper to visualise the mapping result in Fig. 9. Each language feature was assigned a certain colour. Components of the component model were rendered in the color of the feature they are mapped to.

The components `Model Provider` and `TransformationBroker` are mapped to the group feature *Metamodelling Technical Space*. Thus, the are mandatory for every MOST TOPF

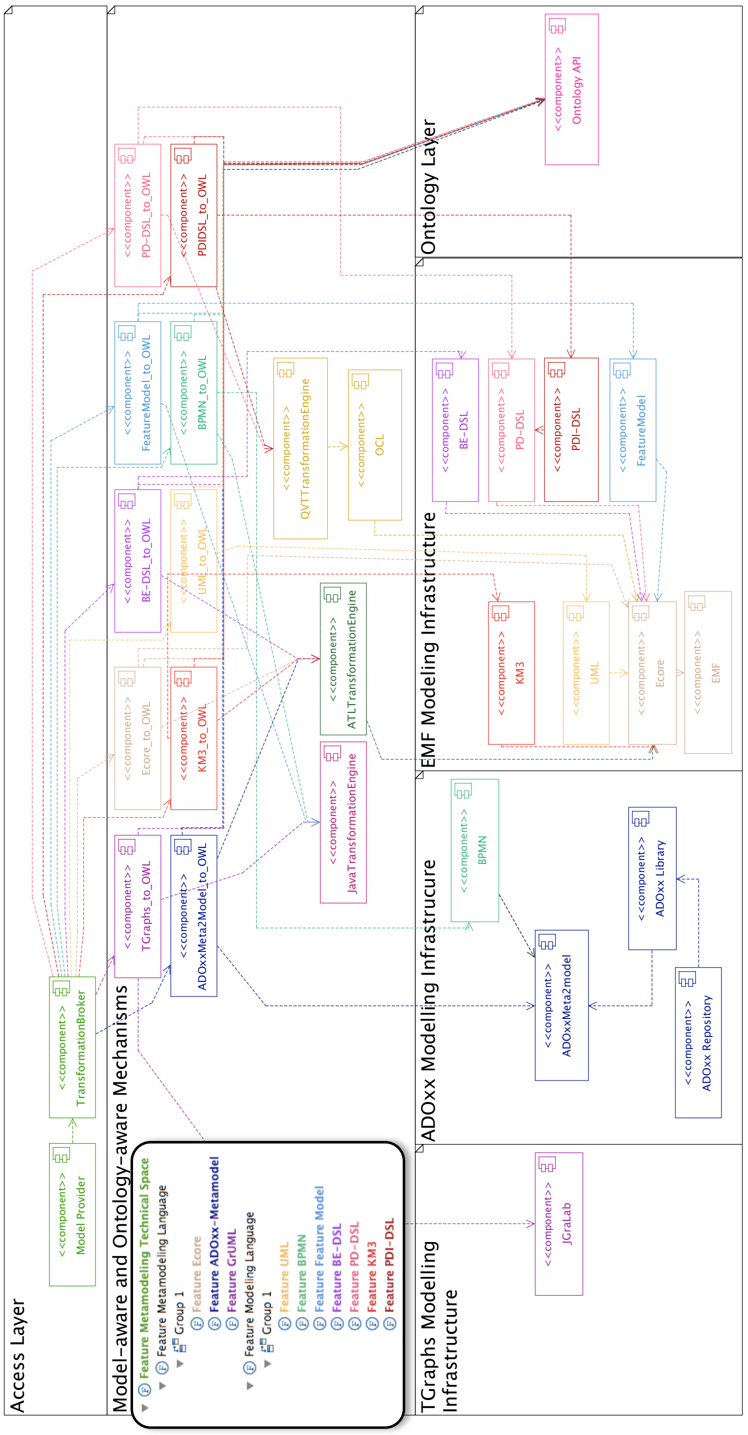

Access Layer
<<component>> Model Provider
<<component>> TransformationBroker
Model-aware and Ontology-aware Mechanisms
<<component>> PD-DSL_to_OWL
<<component>> PDIDSL_to_OWL
<<component>> FeatureModel_to_OWL
<<component>> BPMN_to_OWL
<<component>> BE-DSL_to_OWL
<<component>> UML_to_OWL
<<component>> Ecore_to_OWL
<<component>> KM3_to_OWL
<<component>> TGraphs_to_OWL
<<component>> ADOxxMeta2Model_to_OWL
<<component>> QVTTransformationEngine
<<component>> OCL
<<component>> ATLTransformationEngine
<<component>> JavaTransformationEngine
Feature Metamodeling Technical Space
Feature Metamodeling Language
Group 1
Feature Ecore
Feature ADOxx-Metamodel
Feature GrUML
Feature Modeling Language
Group 1
Feature UML
Feature BPMN
Feature Feature Model
Feature BE-DSL
Feature PD-DSL
Feature KM3
Feature PDI-DSL
Ontology Layer
<<component>> Ontology API
EMF Modeling Infrastructure
<<component>> BE-DSL
<<component>> PD-DSL
<<component>> PDI-DSL
<<component>> FeatureModel
<<component>> KM3
<<component>> UML
<<component>> Ecore
<<component>> EMF
ADOxx Modelling Infrastrucure
<<component>> BPMN
<<component>> ADOxx Library
<<component>> ADOxxMeta2model
<<component>> ADOxx Repository
TGraphs Modelling Infrastructure
<<component>> JGraLab

Figure 9: The mapping of language features to transformation components

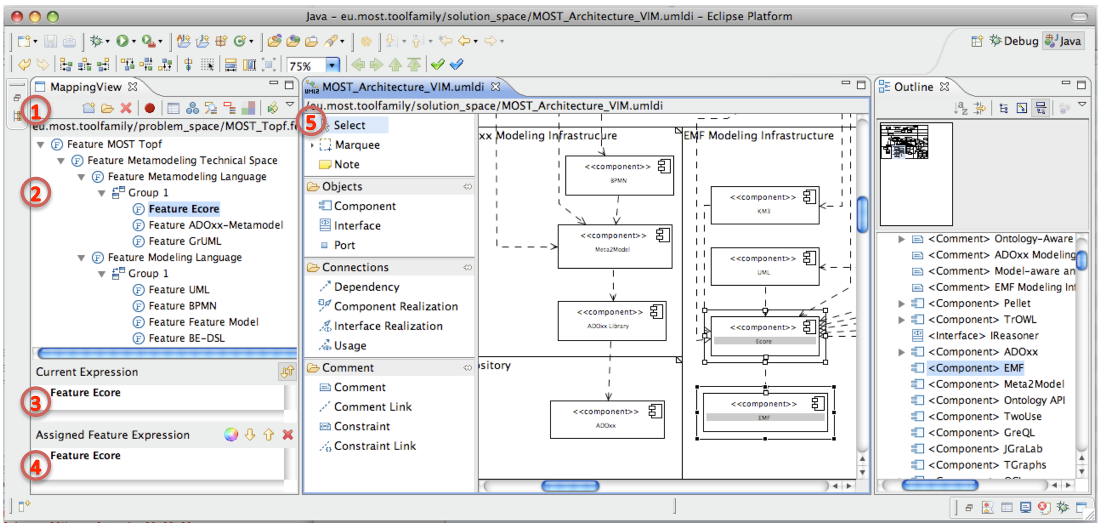


Figure 10: The FeatureMapper is used to interactively map features to elements of a UML components diagram specifying the MOST TOPF architecture

instance. The inclusion of a specific transformation component depends on the selection of the corresponding language. For example consider the language feature `BE-DSL`. It triggers the inclusion of the language component `BE-DSL`, the related dependency relationships, and the transformation component `BE-DSL_to_OWL`.

7.2 Ensuring Dependencies between Transformation Technology Features

In Section 6 we identified several dependencies between features of the technological integration technology for every concrete transformation. To ensure, that these dependencies are satisfied during the configuration of a specific workbench variant model, we included additional constraints in the feature model.

For every language feature a constraint was defined that consists of a conjunction of all features that are required for the technological integration of this language. During variant configuration the FeatureMapper checks this constraint and ensures, that all its required technological features also become part of the variant.

For example the selection of the language feature *PD-DSL* is only valid, if also the features *EMF*, *ADOxx*, *OWL2*, *Transformation*, *QVT*, *Unidirectional*, and *Runtime Time* are part of the variant model. As a consequence, the components these features are mapped to (`Ecore`, `EMF`, `ADOxx Metamodel`, `ADOxx Library`, `ADOxx Repository`, `OWL2`, and `QVTTransformationEngine`) are included for the MOST TOPF product, which results in a valid and complete workbench configuration.

Table 2 summarises all constraints that we derived from the feature dependencies identified in Section 6.

Table 2: Constraints in the feature model to reflect ontology language dependencies

Source Language Feature	**Assigned Constraint**
BPMN	`requires`(EMF `and` ADOxx `and` OWL2 `and` Mapping `and` PlainJava `and` Unidirectional `and` Design Time)
PD-DSL	`requires`(EMF `and` ADOxx `and` OWL2 `and` Transformation `and` QVT `and` Unidirectional `and` Runtime Time)
PDI-DSL	`requires`(EMF `and` PD-DSL `and` ADOxx `and` OWL2 `and` Transformation `and` QVT `and` Unidirectional `and` Runtime Time)
FODA	`requires`(EMF `and` OWL2 `and` Mapping `and` PlainJava `and` Unidirectional `and` Design Time)
UML	`requires`(EMF `and` ADOxx `and` OWL2 `and` Mapping `and` PlainJava `and` Unidirectional `and` Design Time)
Ecore	`requires`(EMF `and` OWL2 `and` Transformation `and` ATL `and` Unidirectional `and` Design Time)
KM3	`requires`(EMF `and` OWL2 `and` Transformation `and` ATL `and` Unidirectional `and` Design Time)
ADOxx Meta2model	`requires`(OWL2 `and` Transformation `and` ATL `and` Unidirectional `and` Design Time)
TGraphs	`requires`(OWL2 `and` Transformation `and` PlainJava `and` Unidirectional `and` Runtime)

7.3 Mapping of the Ontology Technical Space Features

The features of the ontology technical space were also mapped to the corresponding components of the MOST TOPF. This mapping involves the inclusion of components for ontology languages and the reasoning infrastructure. Again, we used the Context View Visualisation of the FeatureMapper to assign colours to workbench components in accordance to the features they are mapped to (cf. Fig. 11 for the mapping results).

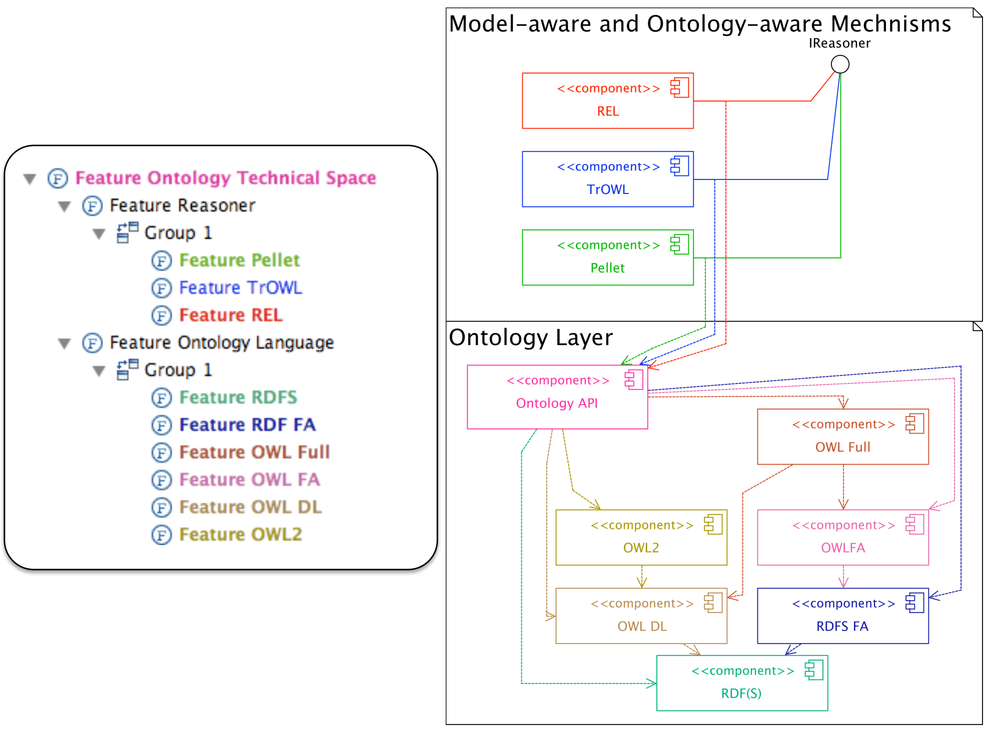


Figure 11: The mapping of ontology technical space features to ontology language components and reasoning infrastructure

The layer of `Model-aware and Ontology-aware Mechanisms` contains a selection of ontology reasoners that are used in MOST. Each reasoner corresponds to a feature from the feature model. In the `Ontology` Layer you find a set of components for the ontology languages used within MOST. As discussed in [Zhao et al., 2008] they form a family of languages which can be organised hierarchically in accordance to their reasoning efficiency and expressive power. More expressive languages like OWL2 subsume and extend less expressive languages. Their language components are organised in accordance to this extension relationship. The access to all ontology language components is managed by a common `Ontology API`.

To reflect the hierarchical organisation during variant configuration, we again used additional constraints in the feature model. These are summarised in Tab. 3.

Table 3: Constraints in the feature model to reflect ontology language dependencies

Ontology Language Feature	**Assigned Constraint**
OWL DL	`requires`(RDFS)
RDFS FA	`requires`(RDFS)
OWL2	`requires`(OWL DL)
OWL FA	`requires`(RDFS FA)
OWL Full	`requires`(OWL FA `and` OWL DL)

8 Conclusion

In this document we illustrated variations on the principle idea of integration the metamodelling technical space (MMTS) with different ontological technical spaces (OTSs). The basic patterns we find in our own work, as well as in related works, is that next to existing technical spaces of established metamodelling frameworks, new technical spaces are positioned that either enrich or exploit the software engineering capabilities by or for ontology technologies. We have identified the main characteristics of such approaches and designed a feature model to enlighten the possible conceptual choices. We have applied our model, illustrating the use of different ontology technologies.

References

[Angele and Lausen, 2004] Angele, J. and Lausen, G. (2004). Ontologies in f-logic. In Staab, S. and Studer, R., editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 29–50. Springer.

[Apostel, 1960] Apostel, L. (1960). Towards the formal study of models in a non formal science. *Synthese*, 12:125–161.

[Atkinson and Kühne, 2003] Atkinson, C. and Kühne, T. (2003). Model-Driven Development: A Metamodeling Foundation. *IEEE Software*, (5):36–41.

[Baader et al., 2003] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., editors (2003). *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press.

[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.

[Berardi et al., 2005] Berardi, D., Calvanese, D., and Giacomo, G. D. (2005). Reasoning on UML class diagrams. *Artif. Intell.*, 168(1):70–118.

[Berners-Lee, 1997] Berners-Lee, T. (1997). Realising the Full Potential of the Web. http://www.w3.org/1998/02/Potential.html.

[Berners-Lee, 1998] Berners-Lee, T. (1998). Semantic Web Road Map. http://www.w3.org/DesignIssues/Semantic.html.

[Bezivin and Kurtev, 2005] Bezivin, J. and Kurtev, I. (2005). Model-based technology integration with the technical space concept. In *Metainformatics Symposium*.

[Bildhauer et al., 2008] Bildhauer, D., Ebert, J., Riediger, V., and Schwarz, H. (2008). Using the TGraph Approach for Model Fact Repositories. In *Proceedings of the 2nd International Workshop on Model Reuse Strategies (MoRSE 2008), published by Fraunhofer IRB Verlag*.

[Bildhauer et al., 2007] Bildhauer, D., Riediger, V., Schwarz, H., and Strauß, S. (2007). grUML–Eine UML-basierte Modellierungssprache für T-Graphen. *Arbeitsberichte aus dem Fachbereich Informatik*, 5.

[Billig et al., 2004] Billig, A., Busse, S., Leicher, A., and Süss, J. G. (2004). Platform independent model transformation based on TRIPLE. In *Middleware '04*, pages 493–511. Springer.

[Brinkkemper, 1996] Brinkkemper, S. (1996). Method Engineering: Engineering of Information Systems Developement Methods and Tools. *Information & Software Technology*, 38(4):275–280.

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

[Calvanese et al., 2005] Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., and Rosati, R. (2005). DL-lite: Tractable description logics for ontologies. In *Proc. of AAAI 2005*, pages 602–607.

[Decker et al., 2005] Decker, S., Sintek, M., Billig, A., Henze, N., Dolog, P., Nejdl, W., Harth, A., Leicher, A., Busse, S., Ambite, J. L., Weathers, M., Neumann, G., and Zdun, U. (2005). Triple - an rdf rule language with context and use cases. In *Rule Languages for Interoperability*. W3C.

[Ebert et al., 2002] Ebert, J., Kullbach, B., Riediger, V., and Winter, A. (2002). GUPRO-Generic Understanding of Programs. *Electronic Notes in Theoretical Computer Science*, 72(2):59–68.

[Frankel et al., 2004] Frankel, D., Hayes, P., Kendall, E., and McGuinness, D. (2004). The model driven semantic web. In *1st International Workshop on the Model-Driven Semantic Web (MDSW2004)*, Monterey, California, USA.

[Friesen et al., 2009] Friesen, A., Lemcke, J., Oberle, D., and Rahmani, T. (2009). D6.2 case studies design. Deliverable ICT216691/SAP/WP6-D2/D/RE/b1, SAP. EU FP7 STREP MOST Project number ICT-2008-216691.

[Happel and Seedorf, 2006] Happel, H.-J. and Seedorf, S. (2006). Applications of ontologies in software engineering. In *International Workshop on Semantic Web Enabled Software Engineering (SWESE'06)*, Athens, USA.

[Heidenreich et al., 2008a] Heidenreich, F., Kopcsek, J., and Wende, C. (2008a). FeatureMapper: Mapping Features to Models. *portal.acm.org*.

[Heidenreich et al., 2008b] Heidenreich, F., Savga, I., and Wende, C. (2008b). On Controlled Visualisations in Software Product Line Engineering. *Proc. SPLC Workshop on Visualization in Software ....*

[Kang et al., 1990] Kang, K., Cohen, S., Hess, J., Novak, W., and Peterson, A. (1990). Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-0211990, Software Engineering Institute.

[Kappel et al., 2006] Kappel, G., Kapsammer, E., Kargl, H., Kramler, G., Reiter, T., Retschitzegger, W., Schwinger, W., and Wimmer, M. (2006). Lifting metamodels to ontologies: A step to the semantic integration of modeling languages. In *Proc. of MoDELS 2006*, volume 4199 of *LNCS*, pages 528–542. Springer.

[Kasztelnik et al., 2009] Kasztelnik, M., Miksa, K., and Sabina, P. (2009). D5.2 case study design. Deliverable ICT216691/CMR/WP5-D2/D/RE/b1, Comarch. EU FP7 STREP MOST Project number ICT-2008-216691.

[Kleppe et al., 2002] Kleppe, A. G., Warmer, J. B., and Bast, W. (2002). *MDA Explained, The Model Driven Architecture: Practice and Promise.* Addison-Wesley, Boston.

[Klyne and Carroll, 2004] Klyne, G. and Carroll, J. J. (2004). Resource Description Framework (RDF): Concepts and Abstract Syntax. http://www.w3.org/TR/rdf-concepts/,. Series Editor: B. McBride.

[Kurtev et al., 2002] Kurtev, I., Bézivin, J., and Aksit, M. (2002). Technological spaces: An initial appraisal. In *CoopIS, DOA'2002 Federated Conferences, Industrial track*, Irvine.

[Mcguinness and van Harmelen, 2004] Mcguinness, D. L. and van Harmelen, F. (2004). OWL web ontology language overview.

[Miller and Mukerji, 2003] Miller, J. and Mukerji, J. (2003). Mda guide version 1.0.1. Technical report, OMG.

[Motik, 2007] Motik, B. (2007). On the properties of metamodeling in owl. *J. Log. Comput.*, 17(4):617–637.

[Motik et al., 2008] Motik, B., Patel-Schneider, P. F., and Parsia, B. (2008). Owl 2 web ontology language: Structural specification and functional-style syntax. World Wide Web Consortium, Working Draft WD-owl2-semantics-20081202.

[Motik and Rosati, 2007] Motik, B. and Rosati, R. (2007). A faithful integration of description logics with logic programming. In Veloso, M. M., editor, *IJCAI*, pages 477–482.

[Oberle et al., 2004] Oberle, D., Eberhart, A., Staab, S., and Volz, R. (2004). Developing and managing software components in an ontology-based application server. In Jacobsen, H.-A., editor, *Middleware*, volume 3231 of *Lecture Notes in Computer Science*, pages 459–477. Springer.

[OMG, 2000] OMG (2000). *Meta Object Facility (MOF) Specification.* Object Modeling Group.

[OMG, 2003] OMG (2003). UML 2.0 superstructure final adopted specification. *OMG Document reference ptc/03-08.*

[OMG, 2005] OMG (2005). *Object Constraint Language Specification, version 2.0.* Object Modeling Group.

[OMG, 2009a] OMG (2009a). *Ontology Definition Metamodel.* Object Modeling Group.

[OMG, 2009b] OMG (2009b). *Unified Modeling Language: Superstructure, version 2.2.* Object Modeling Group.

[Pan and Horrocks, 2007] Pan, J. and Horrocks, I. (2007). RDFS(FA): Connecting RDF(S) and OWL DL. *IEEE Transactions on Knowledge and Data Engineering*, 19:192–206.

[Pan and Horrocks, 2003] Pan, J. Z. and Horrocks, I. (2003). RDFS(FA) and RDF MT: Two Semantics for RDFS. In *Proc. of the 2nd International Semantic Web Conference (ISWC2003).*

[Pan et al., 2005] Pan, J. Z., Horrocks, I., and Schreiber, G. (2005). OWL FA: A Metamodeling Extension of OWL DL. In *Proceeding of the International workshop on OWL: Experience and Directions (OWL-ED2005).*

[Rector et al., 2004] Rector, A. L., Drummond, N., Horridge, M., Rogers, J., Knublauch, H., Stevens, R., Wang, H., and Wroe, C. (2004). OWL pizzas: Practical experience of teaching OWL-DL: Common errors & common patterns. In *Proc. of EKAW 2004*, volume 3257 of *LNCS*, pages 63–81. Springer.

[Roser and Bauer, 2006] Roser, S. and Bauer, B. (2006). An approach to automatically generated model transformations using ontology engineering space. In *Proceedings of Workshop on Semantic Web Enabled Software Engineering (SWESE)*, Athens, GA, U.S.A.

[Silva Parreiras et al.,] Silva Parreiras, F., Staab, S., and Winter, A. TwoUse: Integrating UML models and OWL ontologies. Technical Report 16/2007, University of Koblenz-Landau. Available at `http://isweb.uni-koblenz.de/Projects/twouse/tr162007.pdf`.

[Smith et al., 2004] Smith, M., Welty, C., and McGuiness, D. (2004). OWL Web Ontology Language Guide. http://www.w3.org/TR/owl-guide/.

[Srdjan Zivkovic, 2009] Srdjan Zivkovic, Christian Wende, A. B. (2009). Initial prototype of ontology-driven software process guidance system. Deliverable ICT216691/BOC/WP2-D3/D/PU/b1.00, BOC, Technial University Dresden. EU FP7 STREP MOST Project number ICT-2008-216691.

[Staab et al., 2001] Staab, S., Studer, R., Schnurr, H.-P., and Sure, Y. (2001). Knowledge processes and ontologies. *IEEE Intelligent Systems*, 16(1):26–34.

[Straeten et al., 2003] Straeten, R. V. D., Mens, T., Simmonds, J., and Jonckers, V. (2003). Using description logic to maintain consistency between UML models. In *Proc. of UML 2003*, volume 2863 of *LNCS*, pages 326–340. Springer.

[Tetlow et al., 2006] Tetlow, P., Pan, J. Z., Oberle, D., Wallace, E., Uschold, M., and Kendall, E. (2006). Ontology driven architectures and potential uses of the semantic web in systems and software engineering. W3C Working Draft Working Group Note 2006/02/11, W3C.

[Wang et al., 2007] Wang, H., Li, Y., Sun, J., Zhang, H., and Pan, J. (2007). Verifying feature models using OWL. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):117–129.

[Zhao et al., 2008] Zhao, Y., Pan, J. Z., Jekjantuk, N., Henriksson, J., Gröner, G., and Ren, Y. (2008). De

nition of language hierarchy. Deliverable ICT216691/UNIABDN/WP3-D3.1/D/PU/b1, UNIABDN,TUD,UoKL. EU FP7 STREP MOST Project number ICT-2008-216691.