Project Number: 257909

SPRINT

Software Platform for Integration of Engineering and Things

Collaborative project
Information and Communication Technologies

D3.6 SSI Foundations and Definitions

Start date of project: October, 1st 2010
Duration: 42 months

Deliverable	SSI Foundations and Definitions		
Confidentiality	PU	**Deliverable type**	R
Project	SPRINT	**Date**	2014-02-7
Status	Draft	**Version**	1.0
Contact Person	Michael Wagner	**Organisation**	Fraunhofer FOKUS
Phone	+49 30 3463 7391	**E-Mail**	Michael.Wagner @fokus.fraunhofer.de

THE RESEARCH LEADING TO THESE RESULTS HAS RECEIVED FUNDING FROM THE SVENTH FRAMEWORK PROGRAMME UNDER GRANT AGREEMENT N° 257909

AUTHORS TABLE

Name	Company	E-Mail
Michael Wagner	Fraunhofer FOKUS	Michael.Wagner@fokus.fraunhofer.de

CHANGE HISTORY

Version	Date	Reason for Change	Pages Affected
1.0	2014-02-07	Final Version	All

CONTENT

Content of Figures

Content of Tables

1 Introduction

This document is the final public version of the restricted "D3.1 SSI Foundations and Definitions" to be made public at the end of the project.

1.1 Overview, Purpose and Scope

1.1.1 Purpose of the document

The purpose of this document is to analyse and present suitable and relevant methods and technologies for the definition of the semantic layer of the integration platform of SPRINT. It lays also the foundation for the SSI design decisions in D3.2 Definition of the Semantic Services Integration Layer. In order to do so, an overview of methods, techniques and technologies currently used for semantic analysis and definition is given. This is done by presenting the methods, techniques and technologies and afterward setting them in context. Finally recommendations towards other tasks in the work package will be provided.

1.1.2 Scope of the document

The scope of the document is on the relevant technologies, techniques and methods applicable to the model driven engineering domain.

What is in the scope of the document?

- Relevant meta modelling frameworks in MDE (RDF, EMF)
- Relevant semantic mediation technologies (ontologies, model transformation)
- Relevant semantics and metamodels (HRC, AUTOSAR, EAST-ADL, AADL, and more)
- Relevant integration approaches (Common Meta Model, Transformation Chains, Sea of Information)

1.2 Related SPRINT requirements

ID	Description	
6.1.1	The SPRINT Engineering Environment ontology shall support, and enable the integration of, Engineering Tools from, but not limited to, the following domains: Requirements Engineering, Systems and Software Modelling, Model Simulation, Testing\Validation, Formal Verification, and Contract Authoring	
6.1.3	The SEE shall include a set of Management Tools that will allow for Cross Tool Data Browsing, and will provide Cross Tool Data net views.	
6.1.4	SEE shall support the integration of tools that are deployed using the following paradigms: Client-Server, Web Access, File Based (e.g. Rhapsody)	
6.1.7	The SPRINT platform shall be based on meta-modelling technologies for semantic mediation.	
6.1.8	The SPRINT platform shall adopt standard or de-facto standard technologies wherever available.	
6.1.9	SEE shall support the integration of third party applications (e.g. report generators and model analysis) that can access System Engineering Data, including cross tool and cross domain data, as well as create new data (e.g. system documentation and verification reports).	
6.2.2	Meta-models description shall support standards technologies such as EMF	

ID	Description	
	(Eclipse Modelling Framework), Ecore, Resource Description Framework (RDF) and (Web Ontology Language) OWL.	
6.2.3	The semantic integration of design models shall support a 'light' version of HRC.	
6.2.4	Shared Resources shall have the following services available to them :(1) Queries(2) Administration(3) Storage	
6.2.5	All resources shall be accessible via a URI and have a single owner.	
6.3.2	Simulation tools in SPRINT shall support interleaving semantics for the components execution.	
6.3.3	Design element shall be exchanged between tools by having their resources share the same semantic model.	
6.3.4	Tools shall share the same semantic models for resources of a common understanding when design elements need to be exchanged.	
6.4.1	SPRINT Contract tools (authoring and analysis) shall support CSL (SPEEDS CSL or other agreed contract specification means)	
6.4.6	The integration of physical devices and System designs shall be done using semantic mediation.	
8.2.1	The SEE shall include a service that checks consistency of I\O between a super system and its subsystems.	
8.2.2	The SPRINT Engineering Environment shall include a service that checks the consistency of I\O between models of neighbouring systems (connected subsystems in a super-system).	
8.2.3	The SEE shall include a service that will enable a system engineer to view the relevant Block Diagrams representing the environment(s) in which the system is to be integrated.	
8.2.4	Modelling Tools participating in the SPRINT Project (System Architect, Rhapsody, MathModellica and Simulink) shall each have the capability of representing instances, or objects that are specified on a remote server, and modelled by a different design tool.	
8.2.5	Modelling Tools participating in the SPRINT Project (System Architect, Rhapsody, MathModellica and Simulink) shall each have the capability of creating within the tool, a black box representation of a component that has been modelled, as a black box, using a different tool, based on data available on the SSI\SII.	

Table 1-1: SPRINT requirements considered in this deliverable

2 Methods technology and techniques currently used for semantic analysis and definition

This section gives insight into the methods, technologies and techniques currently used for semantic analysis and definition relevant to the domain of model driven engineering. First the two main different underlying assumptions in modelling are presented as foundation for the understanding of the differences in the techniques and technologies. The second section, presents the relevant modelling frameworks in context with each other, describing commonalities and differences. The third section describes model transformation technologies as approaches for semantic mediation. In the last section general information integration and semantic mediation approaches are presented.

2.1 Assumptions in modelling approaches

In this section two main approaches in modelling are presented. They are evolved in different application areas of modelling and address different issues of modelling reflecting also on the way information is represented and evaluated.

2.1.1 Open World Assumption (OWA)

In the Open World Assumption (CWA) Adding new knowledge or information never falsifies previous assumptions. Assumption is that you don't know everything of relevance and have to explore and incrementally modelled it, without making premature assumptions.

Advantages:

- Easy to extend
- 'representation' of unknown knowledge possible
- Ideal for iterative extension

Disadvantages:

- Hard to retrieve/compute final conclusions/data
- Merge conflicts still possible
- More information to be modelled

2.1.2 Closed World Assumption (CWA)

In the closed world assumption (CWA), at any given instance, everything that is not known to be true is false. This is a simplification in the evaluation in conjunction with the assumption that the information basis is complete. Assumption is that you know everything of relevance and have modeled it, which is very prominent in the model driven engineering domain.

Advantages:

- Easier to compute
- Quicker to model (less information needed)
- WYSIWYG (no unexpected additions)

Disadvantages:

- Limited expressiveness (“no modelling of the unknown”)
- Forgetting something leads and later insertion leads to wrong/new/non-conform results
- Added new knowledge or information can lead to the falsification of previously made assumptions.

2.1.3 Example on OWA vs. CWA

In this example we consider the available knowledge modelled at a certain point in time, a query and the given answer considering the evaluation mechanism of the two assumptions.

	CWA	**OWA**
Knowledge	Michael is Swimmer Christian is Swimmer	Michael is Swimmer Christian is Swimmer
Query	Is Sandra Swimmer	Is Sandra Swimmer
Result	NO!	Unknown!

Table 2-1: OWA vs. CWA Example Basis

In the basic example the same knowledge and query exist for both approaches but the results are different. This is due to the different interpretation of data. In CWA knowledge which is not available is assumed as not existing. This results in the evaluation of the example to “NO” in contrast to OWA where the result is “Unknown” since facts not explicitly stated are interpreted as unknown.

	CWA	**OWA**
Knowledge	Michael is Swimmer Christian is Swimmer Sandra is Swimmer	Michael is Swimmer Christian is Swimmer Sandra is Swimmer
Query	Is Sandra Swimmer	Is Sandra Swimmer
Result	NO -> YES **(CONTRADICTION)**	Unknown -> YES! Valid change.

Table 2-2: OWA vs. CWA Example contradictory change

In this example you can see a contradictory change simply by adding knowledge. In CWA this added knowledge has the effect that the result changes from “NO” to “YES” which is contradictory whereas in the OWA the change from “Unknown” to “YES” is conform.

	CWA	**OWA**
Knowledge	Michael is Swimmer Christian is Swimmer	Michael is Swimmer Christian is Swimmer Sandra is Non-Swimmer Non-swimmer and Swimmer are disjoint
Query	Is Sandra Swimmer	Is Sandra Swimmer
Result	NO!	Unknown -> NO! Valid change.

Table 2-3: OWA vs. CWA Example conformant change

The example conformant change describes how a “NO” result can be achieved with the OWA.

2.2 Modelling frameworks

The aim of this chapter is to give an overview of available modelling frameworks their commonalities and differences. This is done by giving a short motivation for modelling and meta-modelling. After this SSI relevant modelling frameworks are presented and described. Later, these modelling frame works are set in context in order to present the commonalities and differences.

In the area of systems development modelling is used in order to shift complexity and to focus only on particular aspects of information. This makes it usually easier to solve problems for this aspect. Another benefit of a model is that, in contrast to the real world, things are more easily undoable and usually not so expensive.

Meta modelling allows us to generically describe information about the model. This includes for example the information on the structure of the information been modelled. (See Figure 2-1: Meta modelling and layers MOF example) This mechanism allows data to be more generically utilized or reused since the structure of the data can be explored without prior knowledge of the content. This allows that a constraint checker can be applied to multiple models (M1) with different metamodels (M2) without adaptation since the checking rules

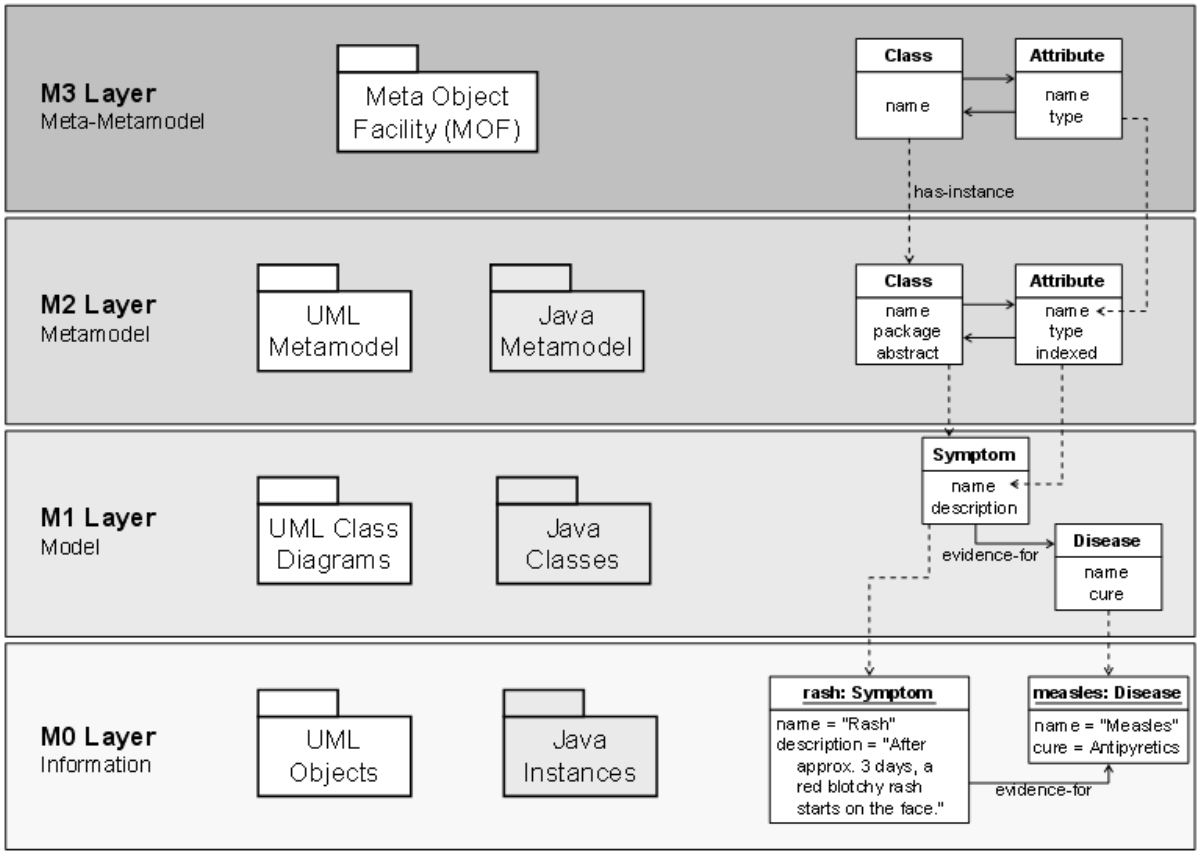

Figure 2-1: Meta modelling and layers MOF example[1]

[1] Source: http://protege.stanford.edu/plugins/xmi/omg-metamodels.png

In this document we assume always modelling in conjunction with meta-modelling if not stated otherwise.

2.2.1 Properties of metamodel elements

This section describes possible properties of metamodel elements utilized for later classification of the different frameworks.

2.2.1.1 Independence (first class elements)

Independence means that the metamodel element can exist and define something without requiring another element.

Example:

Classes are usually fist class metamodel elements since they define a semantic by their own and do not need to relate to other elements.

2.2.1.2 Derivation

Derivation is the case if something can be declared as to be calculated out of other information.

Example:

Age of a person is 20. If he is adult can be computed out of his age (age >= 18).

2.2.1.3 Disjoined

Indicates that instances ore elements associated with this metamodel element are not of associated the same way with another metamodel element the disjoint relation is established with.

Example:

Adult and juvenile persons are two disjoined sets.

2.2.1.4 Final

Here final has the meaning of not being able to be changed or extended.

Example:

Class which is not allowed to be further inherited.

2.2.1.5 Ordered

Ordered means that relations to this object can maintain an order over their lifetime. This represents additional information and can be utilized in interpretation. Lack of such feature can lead to additional modelling data overhead if required.

Example:

UML operation parameters: add(list : List, element : Object)

2.2.1.6 Unique

Unique means that relations to this object can maintain uniqueness over their lifetime. This poses an additional constraint.

Example:

Sebastian friends -> (Susanne, John, and Mark)

2.2.1.7 Multiplicity

Multiplicity is a special kind of constraint, it constraints the number of related Object via a given property.

Example:

Person biologicalParent [2..2]

2.2.2 Features of meta-metamodels

2.2.2.1 Typing

Typing means that you support a type system (in this case static typing). A type system is a tractable syntactic framework for classifying phrases according to the kinds of values they compute.

Example (typed variable definition and initialization):

int x = 5:

2.2.2.2 Instantiation

Instantiation means that something belongs to a certain kind of classification. It can additionally mean that the classification expresses properties of the structure of the instance.

Instantiation is often in relation to a type concept.

Example:

Mark : Person

Mark <<instanceOf>> Person

2.2.2.3 Properties

Properties express characteristics of an object.

Example:

(Age of) Mark is 14.

2.2.2.4 Relation

Relations represent the capability to set two objects (or the same) in relation with each other.

Example:

Tom is the father of Mark.

2.2.2.5 Hierarchy

Hierarchy allows the organization of elements in a hierarchical way. This has usually also influence on the way of how elements can be addressed.

Example:

UML: Pachage -> Class -> Subclass -> Property

2.2.2.6 Containment

Containment is a feature of lifecycle management. It expresses that every (except special root elements) model elements have to be contained somewhere. The underling thinking is that model elements do not just exist by their own. Containment has also a consistency capability that an element cannot contain itself.

Example:

Germany -inhabitants> Klaus,Meyer –owns> Hous –has> Chimney

2.2.2.7 Inheritance

A way of compartmentalize the reuse of structural (and behavioural) definitions. The inheritance relationship is transitive.

Example:

class B : public A {};

2.2.2.8 Constraints

Constraints allow the definition of restrictions.

Example (OCL):

context Vehicle inv: self.owner.age >= 18

2.2.3 Extensible Markup Language (XML) and XML Schema XSD

Extensible Markup Language (XML) is a language for the definition of hierarchical structures with textual representation. XML Schema XSD is the language for defining these structures. They are mentioned here for completeness since the focus of the documents is on the semantic level and these languages are on the structural.

2.2.4 Meta Object Facility (MOF)

The extensive application of models and modeling techniques within the scope of the development lead to the requirement for a unique and standardized framework for the management, manipulation and exchange of models and meta-models. This need has been addressed by the OMG with the Meta-Object-Facility (MOF). The Meta Object Facility (MOF) is the definition of a platform-independent metadata framework.

2.2.4.1 Layers

The MOF standard [MOF, 2002] specifies a framework for managing arbitrary metadata. It defines a modelling architecture as shown in Figure 2-2: MOF Layers. The architecture of MOF is based on the traditional four-layer approach of meta-modelling:

- Layer M0 - the instances: information (data) that describe a concrete system at a fixed point in time. This layer consists of instances of elements of the M1-layer.
- Layer M1- the model: definition of the structure and behaviour of a system using a well-defined set of general concepts. An M1-model consists of M2-layer instances.
- Layer M2 - the meta-model: The definition of the elements and the structure of a concept space (i.e. the modelling language). An M2-layer model consists of instances of the M3-layer.
- Layer M3 - the meta-meta-model: The definition of the elements and the structure for the description of a meta-model.

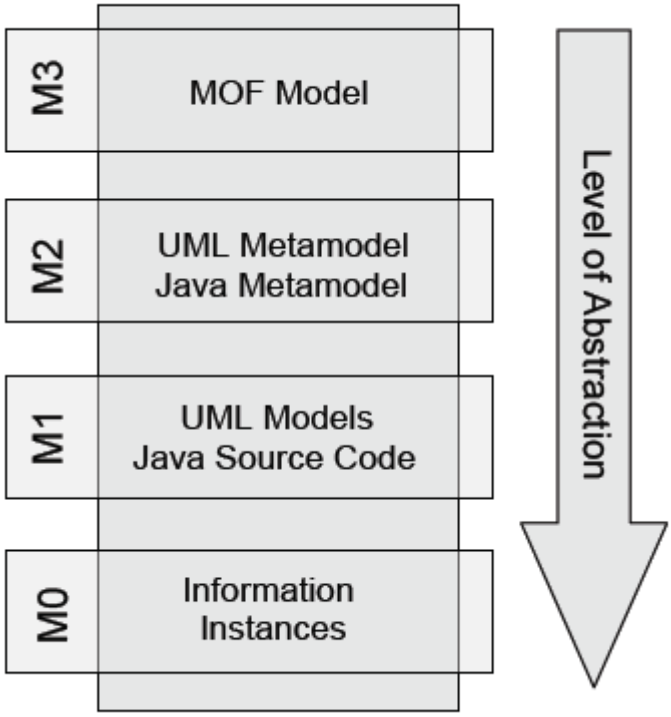

Figure 2-2: MOF Layers

The Level M0 accommodates information which can be structured in an object oriented or non-object oriented manner. It is the lowest level of abstraction in this four-level architecture. The objects metadata, described in (domain) models, belongs to the M1 level. As models itself must be defined, there are means to describe them. Those modeling languages (meta-models) reside on the M2 level. Last but not least, the object oriented constructs for specifying meta-models are building the M3 level meta-meta-model.

2.2.4.2 Constructs

The advantage of the MOF-approach is to make the definition of models and meta-models independent from the concrete application domain of the models and to provide a concise and unique set of concepts for the definition of meta-models. By this the MOF can manage multiple meta-models.

Elements and structure of the M3-layer are directly derived from a well-known formalism, in case of MOF object-orientation. In fact, the MOF concepts are defined using MOF itself. The MOF-model consists of the following concepts for the definition of meta-models:

- **Classes** are type definitions for objects at the M1 level. Those Class instances have identity, state and behavior. Attributes are the structural features and describe state, as behavior is represented by operations. In addition, a Class can generalize other Classes for hierarchical decomposition by means of inheritance. A Class marked as abstract is not allowed to have instances.
- Binary Relationships between Classes are defined by **Associations** with links as their instances. Links do not have identity or state. The only features of Associations are AssociationEnds. Aggregation semantics of Associations can be specified by the corresponding AssociationEnd property supporting the values composite and non-aggregate. To support an Attribute-like view to Associations, MOF allows the definition of References which encapsulate the knowledge of associated Classes.
- **Datatypes** specify complex types whose instances have no identity.
- **Packages** are used to organize meta-models. They can contain all modeling constructs including other Packages. Applicable mechanisms here are nesting and import.

The MOF specification defines these concepts as well as supporting concepts in detail. Because there is no explicit notation for MOF, the UML notation has been deliberately used to visualize selected concepts.

The remaining modeling elements are Constraints, Tags and Constants. As the standard does not define any means for the definition of model element constraints, the MOF model itself is using OCL extensively. Tags allow the annotation of information to MOF model elements without typing. A Tag is a key/value pair which supports the storage of a string value under a specific name. The last construct to mention are MOF Exceptions. Modelers can declare them to be raised by an Operation.

2.2.4.3 History

The MOF 1.4 standard, published in April 2002, was replaced by the major revised version MOF 2 in January 2006 [MOF, 2006]. The most remarkable changes have been made to the structure of the MOF model itself. One anticipated goal for the standard definition was a better alignment of MOF and UML. To achieve this goal, both standards share the same core. A UML subset forms the UML 2 Infrastructure Library [UMLI, 2003] which is reused by both, MOF 2 and UML 2. Furthermore, the MOF Model is structured in reusable packages and two bigger parts called Essential MOF (EMOF) and Complete MOF (CMOF). This distinction was introduced to allow tool vendors easier being compliant to MOF. EMOF provides constructs for the definition of simpler meta-models and supports extensions. Figure 2-3: MOF Overview and imports from UML Core [MOF2] shows which packages are used by EMOF. CMOF is intended to be used for building more complex meta-models like the UML 2 meta-model.

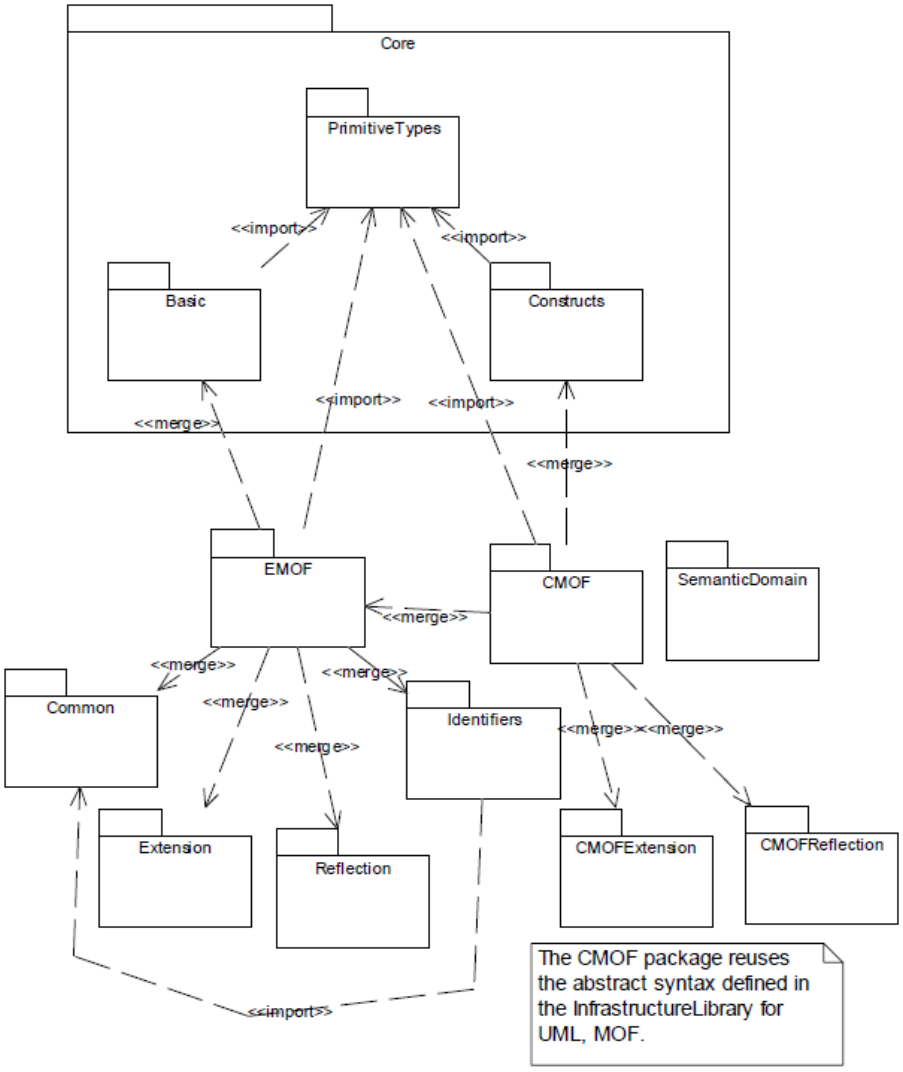

Figure 2-3: MOF Overview and imports from UML Core [MOF2]

How Many "Meta Layers"?

MOF 1 and MOF 2.0 allow any number of layers greater than or equal to 2. (The minimum number of layers is two so we can represent and navigate from a class to its instance and vice versa). Suffice it to say MOF 2.0 with its reflection model can be used with as few as 2 levels and as many levels as users define. [MOF2]

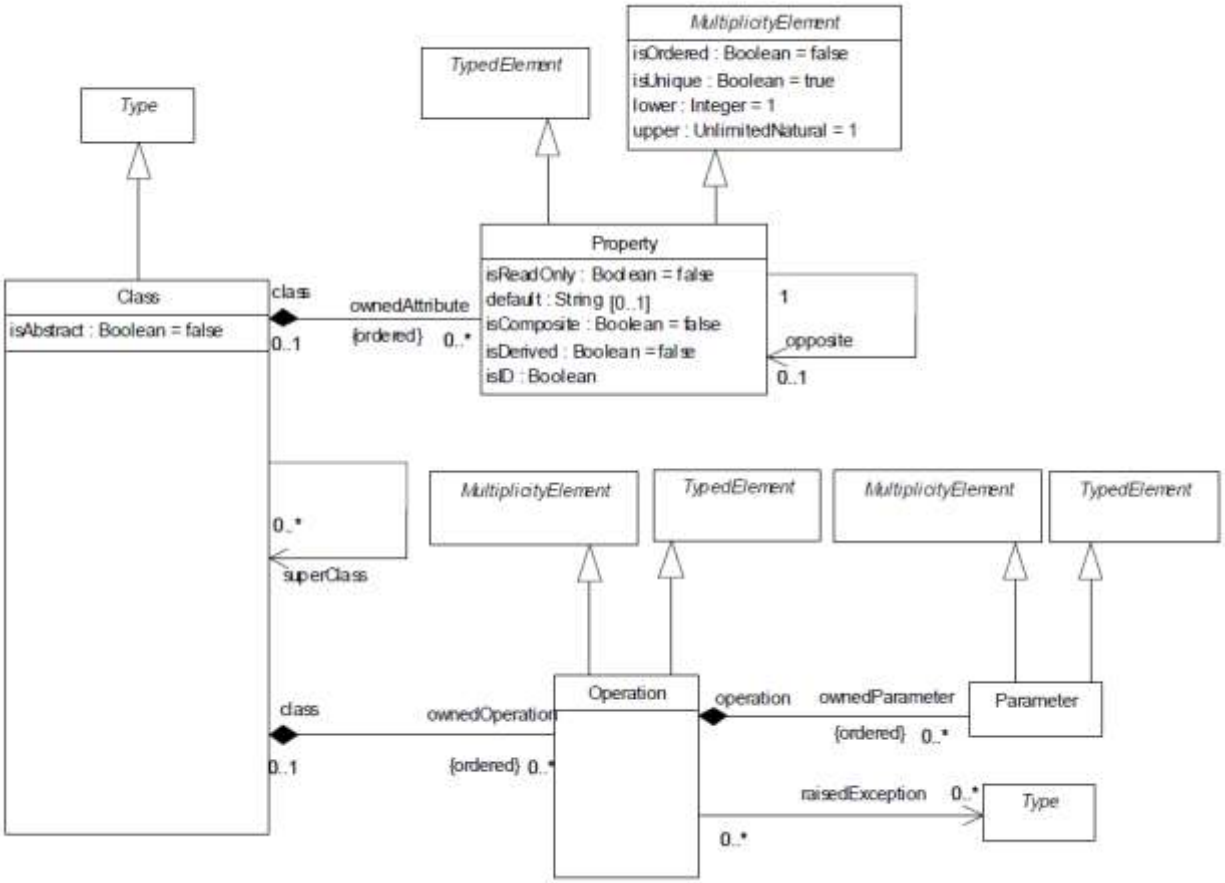

Figure 2-4: EMOF Classes [MOF2]

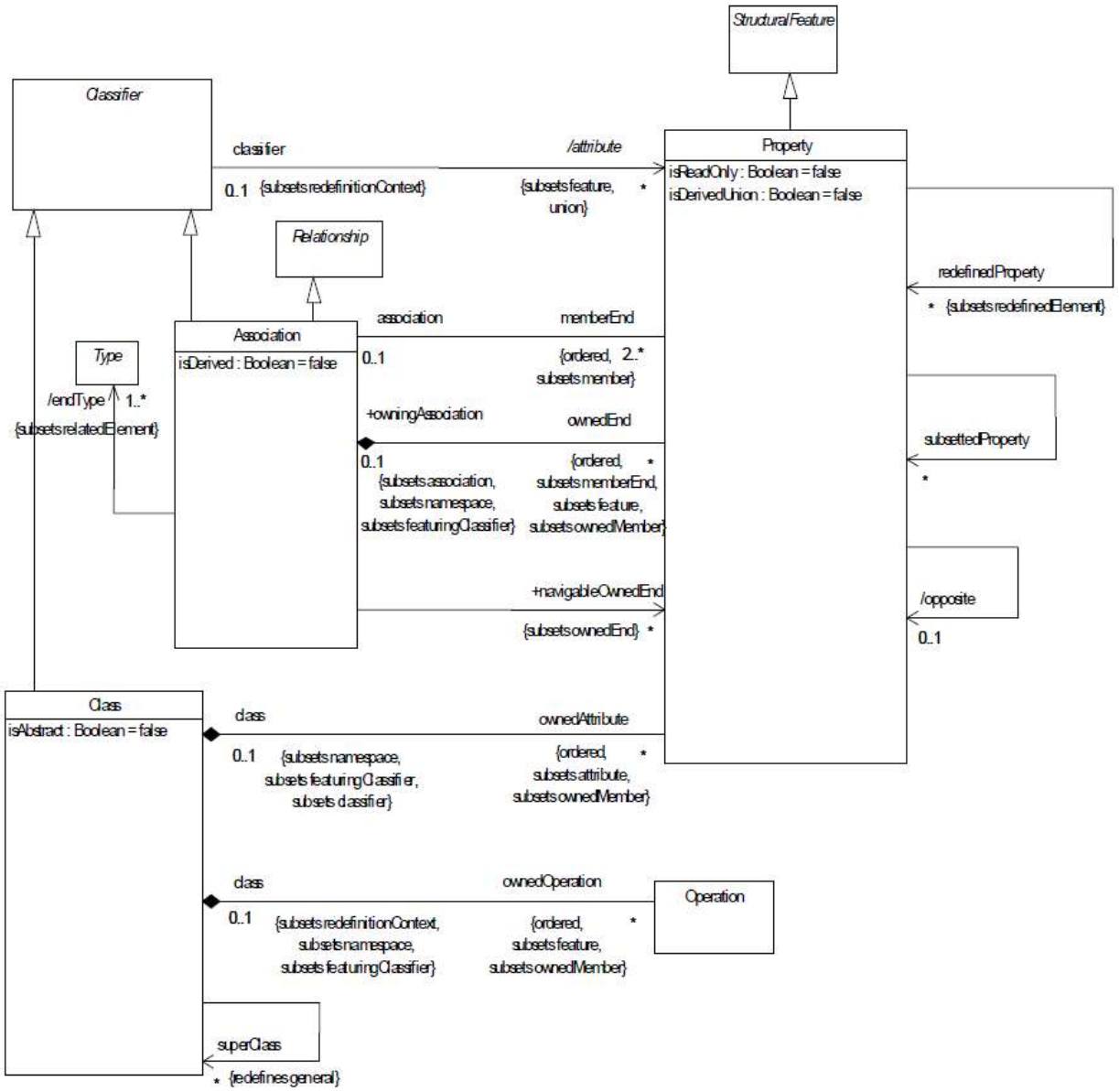

Figure 2-5: CMOF Classes [MOF2]

2.2.4.4 Mappings

The Abstract Mapping is also part of the MOF standard and defines how instances of MOF based meta-models can be created in a certain technology context. It is used to define mappings from MOF to the Interface Description Language (IDL), the Java Metadata Interface (JMI, [JSR040, 2002]) and XMI. With the help of CORBA IDL or JMI it is possible to automatically create model repositories from MOF compliant meta-models. Such repositories are used to store and manage meta-model instances, i.e. models.

With these standardized technologies, MOF proofs its importance for MDA. The generated model repositories support the interoperability among several tools, as they provide a standardized interface for accessing models. Meta-model instances can be represented in XMI and exchanged among several software applications. Figure 6: Tool Integration in MDA Environment depicts a typical MDA tool chain.

XML Metadata Interchange (XMI) is an OMG standard that maps the MOF to the W3C's eXtensible Markup Language (XML). XMI is a standard interchange mechanism used between various tools, repositories and middleware; it defines how XML tags are used to represent serialized MOF-compliant models in XML. MOF-based meta-models are translated to XML Document Type Definitions (DTDs) and models are translated into XML Documents that are consistent with their corresponding DTDs. XMI has been used to render UML artefacts (using the UML XMI DTD), data warehouse and database artefacts (using the CWM XMI DTD), CORBA interface definitions (using the IDL DTD), and Java interfaces and Classes (using a Java DTD).

XMI, which marries the world of modeling (UML), metadata (MOF and XML) and middleware (UML profiles for Java, EJB, CCM, EDOC etc.) plays a pivotal role in the OMG's use of XML at the core of the MDA. In essence XMI adds Modeling and Architecture to the world of XML.

2.2.4.5 Eclipse modelling Framework EMF

The EMF is a java based implementation of EOMF and widely in use in the model driven community. Figure 2-6: EMF Overview shows the equality to EMOF.

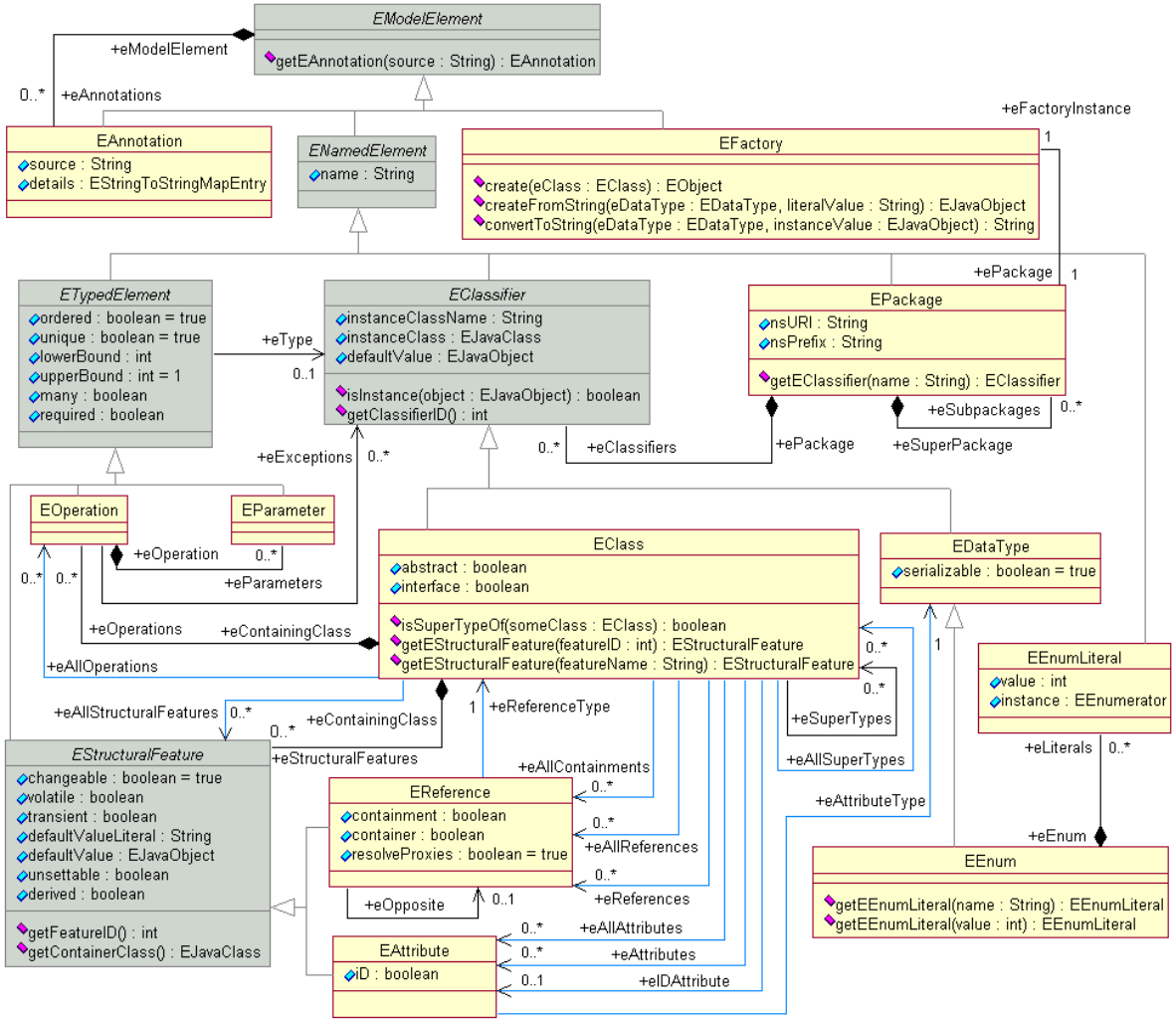

Figure 2-6: EMF Overview

2.2.5 Object Constraint Language (OCL)

MOF and UML diagrams are typically not refined enough to provide all the relevant aspects of a specification. There is, among other things, a need to describe additional constraints about the objects in the

model. Such constraints are often described in natural language. Practice has shown that this will always result in ambiguities. In order to write unambiguous constraints, so-called formal languages have been developed. The disadvantage of traditional formal languages is that they are usable to persons with a strong mathematical background, but difficult for the average business or system modeler to use. The Object Constraint Language (OCL) has been developed to fill this gap. It is a formal language that remains easy to read and write. It has been developed as a business modeling language within the IBM Insurance division. OCL is a pure specification language; therefore, an OCL expression is guaranteed to be without side effects. When an OCL expression is evaluated, it simply returns a value. It cannot change anything in the model. This means that the state of the system will never change because of the evaluation of an OCL expression, even though an OCL expression can be used to specify a state change (e.g., in a post-condition). OCL is not a programming language; therefore, it is not possible to write program logic or flow control in OCL. You cannot invoke processes or activate non-query operations within OCL. Because OCL is a modeling language in the first place, OCL expressions are not by definition directly executable.

2.2.5.1 Constraint context and kind

The OCL allows software developers to write constraints and queries over object models. These constraints are useful, as they allow a developer to create a highly specific set of rules that govern the aspects of an individual object. Each constraint is defined within a context. A context can be a:

- Class: a MOF or UML class
- Attribute: an attribute of a MOF or OCL class
- Operation: an operation of a MOF or OCL class

These constraints have also a kind which defines the character of the constraint. In OCL there exists seven ConstarindKinds which are:

1. Pre: is used to define the pre-conditions of an operation
2. Post: defines the post condition of an operation. The '@pre' operant allows you to access the pre execution state of the model to define the post state.
3. Body: can be used to describe semantic of query operations – query operations are operations specially marked as query within the metamodel.
4. Inv: invariants are used to describe consistency rules over model elements.
5. Init: defines the initial value for the attribute of an object.
6. Derive: is used to describe in which manner the value for an attribute of an object is derived from the other values of this object.
7. Def: definitions can be used to define additional helper attributes and operations to already existing classes. This increases the reuse of OCL expressions and readability.

2.2.5.2 Types in OCL

OCL is a typed language. Besides the OclModelElement types defined by the current meta-model OCL predefines own types (see Figure 8: Types defined in OCL standard library). Most important of these are:

1. OclAny: can be compared with java.lang.Object of the Java language. It is the basic object all besides collections inherit from. OclAny defines important operations like oclIsOfType, oclIsOfKind, oclAsType, '=' and '<>'.
2. OclVoid: is the type which conforms to all other types. OclVoid has only one instance called OclUndefined. OclUndefined represents error value. Even if OclUndefined could be compared with

the Java null it is more like a java.lang.Exception. Since every access to OclUndefined results in OclUndefined itself – with the exception of the Boolean operators. It is possible to avoid further computation on OclUndefined by using the isOclUndefined operation.

3. Primitive types: OCL defines also some primitive types. The range of these types – for example Integer – depends on the chosen metamodel. For example the range of Integer in MOF is 32 bit twos-complement to support limited implementations.
    a. Real: a Real represents a floating point value.
    b. String: represents an atomic string since OCL has no support for characters. This might also one of the causes to remove operations like toUpper and toLower (Case) from the standard.
    c. Boolean: boolean value – since OclVoid conforms to Boolean it is somewhat of an tree state boolean value.
    d. Integer: signed integer type
4. Collection: in contrast to the other types Collections do not inherit from OclCollection. Collections share a common set of operations like size and empty. The fact that collections in OCL are templates is one of the main problems implementing the OCL standard. The four supported set types represent different combinations of uniqueness and order:
    a. Set: unordered unique
    b. OrderedSet ordered unique
    c. Bag unordered not unique
    d. Sequence ordered not unique

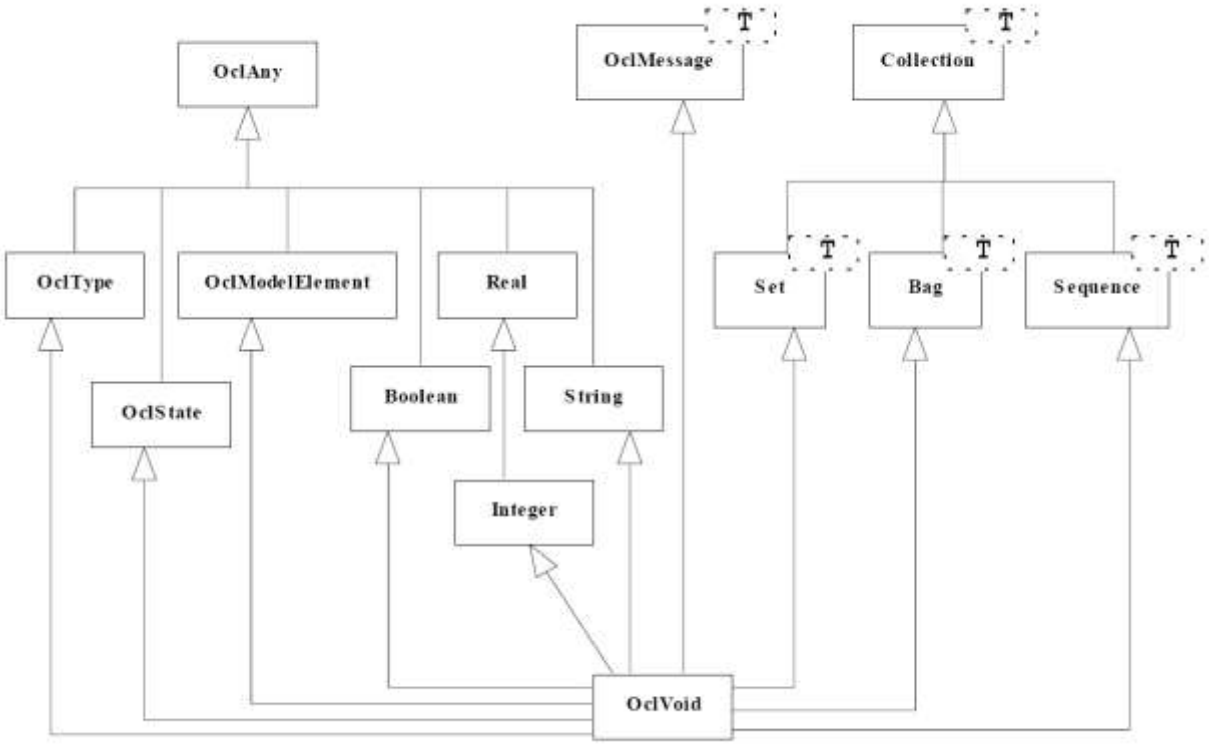

Figure 2-7: OCL Types

2.2.5.3 Notation and Examples

OCL uses a textual notation for the formulation of constraints. Since attempts for a visual notation have clearly shown that this is not practicable. As usual for textual notation languages OCL defines keywords:

and, attr, context, def, else, endif, endpackage, if, implies, in, inv, let, not, oper, or, package, post, pre, then, xor

It is possible to escape identifiers using the underline as prefix to avoid name clashing of keywords with meta-model elements. All constraints start with the keyword constraint, the context model element and the constraint kind. Followed by the defining OclExpression.

constraint <MyModelPackage::MyModelElement> <ConstraintKind>: <OclExpression>

For example to state that all persons within the systems have to be over 17 would look like this:

constraint Person inv: age > 17

This is a good example for an invariant. The same expression as a derived attribute would be represented this way:

constraint Person::adult : Boolean derive: age > 17

2.2.6 Query View Transformation (QVT)

Besides MOF, QVT is an important standard within the MDA. As demanded in [QVTRFP, 2002], the standard must address the definition of model queries, model views and model-to-model transformations. Queries are used to retrieve specific elements from a model in an ad-hoc manner to be able to process them afterwards. For accenting parts of a model, views can be defined. Therefore, one model is deduced from another model and unwanted model parts are hidden. In other words, a view is a read only transformation [Koehler, 2003].

The QVT standard [QVT, 2005] deals with the standardization of model-to-model- transformations. Queries can already be defined by OCL and if there is a possibility to define model transformations, they can be used to specify views as they are read only transformations. While the standard is still work in progress it has reached the state of a Final Adopted Specification. In fact, the normative document results from a combination of a list of submitted proposals.

The QVT standard defines both, declarative and imperative approaches for model transformation definitions on the M2 meta-level (see Figure 2-2: MOF Layers). According to [Torgersson, 1996], declarative languages focus on the definition of relationships between elements and the language processor applies a fixed algorithm to produce a result. The system state is changed once from the user's viewpoint. In contrast, an imperative language provides constructs to explicitly define multiple changes of the system state. In other words, the first defines what has to be changed and the latter how changes are computed. In fact, the QVT standard does not define a fixed algorithm for processing the element relationships. Instead, fixed semantics are specified for them. The realization is left to the implementers.

Several partly dependent metamodels, which partly depend on each other, encapsulate the different techniques. Figure 2-8: QVT Package Overview depicts the relationships to the MOF and OCL standards as well as dependencies among the QVT packages. On one hand, the QVT meta-model is based on the EMOF package. On the other hand, it relies on the Essential-OCL package which provides a small OCL fundament,

necessary to work with EMOF. Besides the QVT-Core, QVT-Relation and QVT-Operational packages which encapsulate the three language parts of QVT, some additional ones are defined:

- QVT-Base provides a common structure for transformations.
- The QVT-Template package is used by QVT-Relation for the definition of template pattern expressions.
- Imperative expressions for QVT Operational Mapping are defined in Imperative OCL

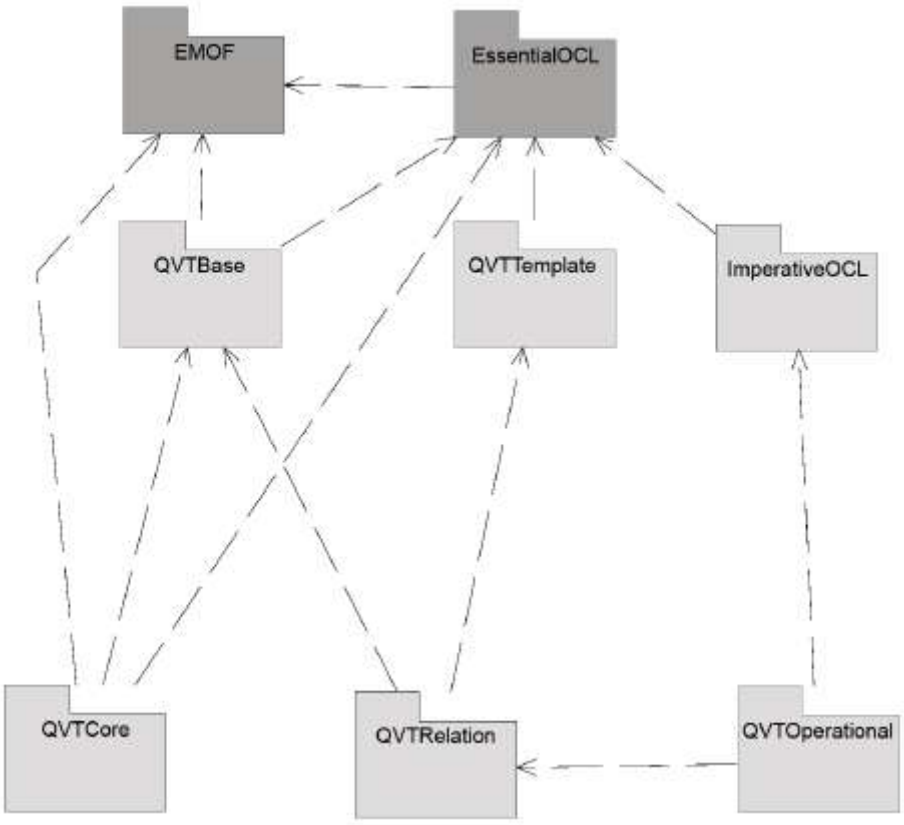

Figure 2-8: QVT Package Overview

The declarative parts are organized in two abstraction levels. In opposite to the lower level QVT Core, QVT Relations is more suited for end users by providing a higher level, purely declarative transformation definition language, which is based on template pattern expressions defined in the QVTTemplate package. As can be seen in Figure 2-8: QVT Package Overview, QVT Core directly uses the elements of the EssentialOCL package.

While it introduces little extensions to EMOF, QVT Core forms a small language for transformations which is directly based on EssentialOCL. It can either be used to implement transformations directly, or to provide formal semantics for QVT Relations, because both have the same expressiveness. Therefore, the standard defines a transformation from Relations to Core. As the language itself is very simple and text based only, it is easier to describe its semantics but transformation descriptions are less user friendly. Implementers must take care of traces by themselves and the pattern definition for object matching only relies on flat variables. More detailed description of can be found in section 2.3.2 QVT.

2.2.7 Resource Description Framework (RDF) and RDF-Schema (RDFS)

The Resource Description Framework (RDF) and RDF-Schema (RDFS) is a descriptive language for the definition of structured vocabularies defined by the W3C. It is assembled out of subject, predicate and object triples. These are assembled to describe higher concepts like classes, types, properties, restrictions and hierarchical relationships like subclass and subproperty relationships (See: Figure 2-9: RDF and RDFS Classes Overview).

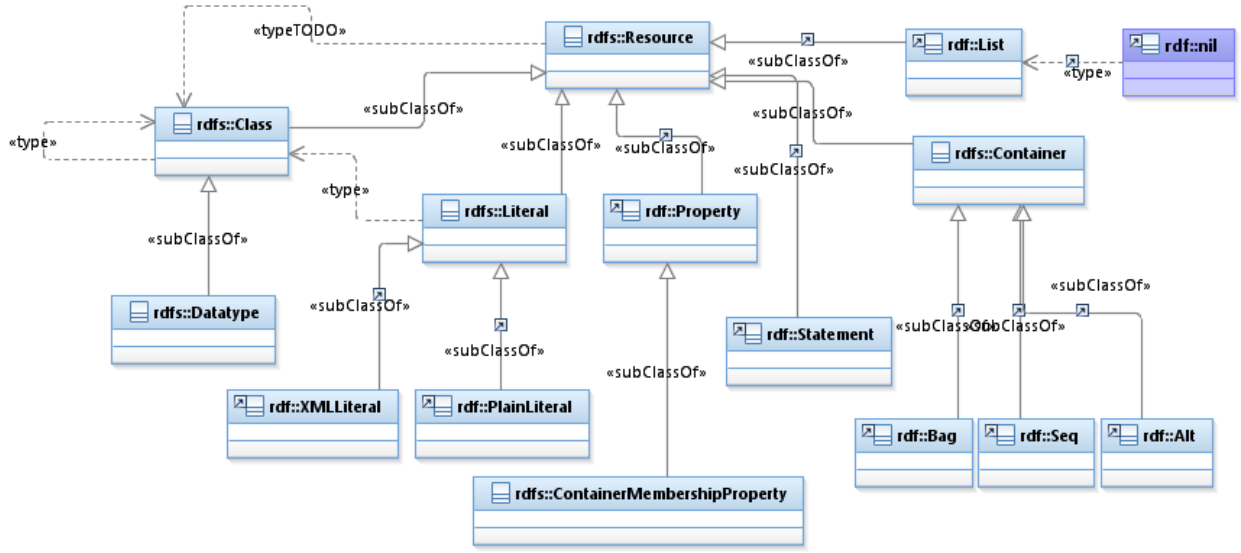

Figure 2-9: RDF and RDFS Classes Overview (UML Class Notation)

RDF/RDFS main language class concepts:

- **rdfs:Resource** : Every entity in a RDF model is instance of this class.
- **rdf:Property** : Basic super class for all properties to be defined.
- **rdfs:Class** : Class concept which defines an abstract object and in combination with rdfs:Type instance relationships.
- **rdfs:Literal** : Class for literal values (e.g. String)

(Additional: rdfs:Datatype, rdfs:XMLLiteral, rdfs:Container, rdfs:ContainerMembershipProperty)

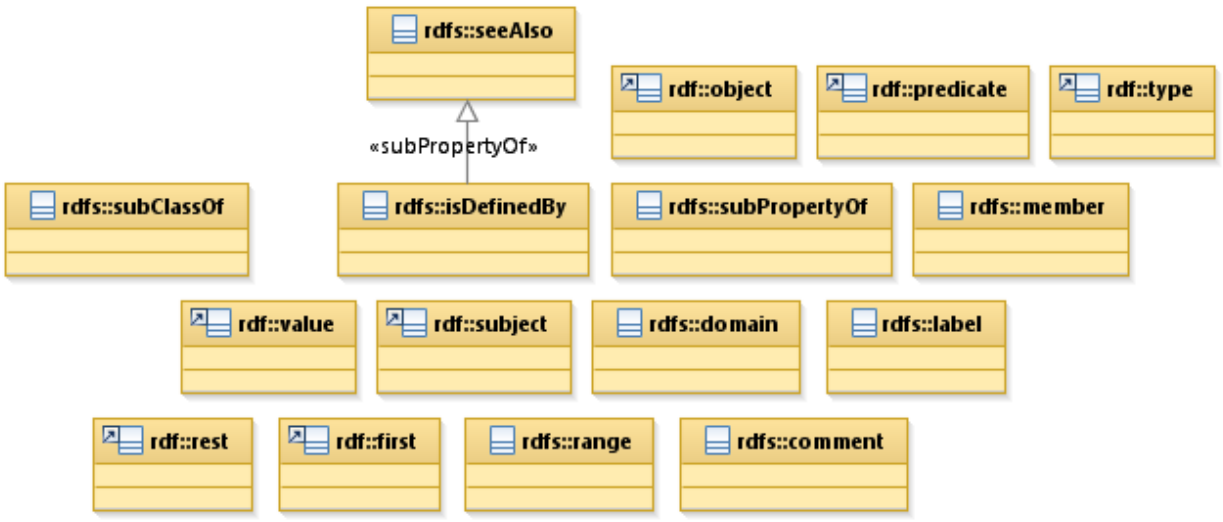

Figure 2-10: RDF and RDFS Properties Overview (UML Class Notation)

RDF/RDFS main language property concepts:

- **rdfs:subClassOf** : Transitive property for the definition of class inheritance hirarchies.
- **rdfs:subPropertyOf** : Transitive property for the definition of property inheritance hirarchies.
- **rdfs:Domain** : Definition of the application area of a property in relation to a class.
- **rdfs:Range** : Defines the allowed range of values for a given property.

All these concepts allow the structuring of information. Hoverer, even if there is a property range defined. It is with RDF/RDFS not possible to describe constraints on model elements.

Turtle is a format which allows easy human readable textual representation of RDF triplets in contrast to RDF/XML.

2.2.8 SPARQL Protocol And RDF Query Language (SPARQL)

The SPARQL Protocol And RDF Query Language (SPARQL) can be utilized for extraction, discovery, complex join operations, transformations, construction of new RDF graphs. The language is based on turtle a textual representation of RDF triplets. This representation is extended with variables in order to define triple patterns for query matching.

Important language constructs:

- **Variable:** A variable specifies an unbound element for matching. Blank notes are treated as variables which cannot be selected. Variables are denoted through a starting '?' or '$'.
- **Triple Pattern:** A triple pattern is represented by an RDF triple containing variables.
- **Graph Patterns:** Graph patterns are combined triple patterns with a special semantic:
  - **Group Graph Pattern:** The group pattern represents a conjunction of the patterns where all variables must match for all triple patterns.
  - **Optional Graph Pattern:** The group pattern represents a conjunction of the patterns where all variables should match if possible. The match is still valid even if the optional pattern part does not match.
  - **Union Graph Pattern:** The union graph pattern creates a union from the results of respective single patterns.
- **Filter:** Allows filtering of results.
- **Ordering:** Allows the definition of the order of the results.
- **Limit and Offset:** Allows the definition of a range for the result.
- **Construct:** Allows the construction of new triples from query results.

Subject	predicate	Object
a	foaf:name	"Johnny Lee Outlaw"
a	foaf:mbox	<mailto:jlow@example.com>
b	foaf:name	"Peter Goodguy"
b	foaf:mbox	<mailto:peter@example.org>
c	foaf:mbox	<mailto:carol@example.org>

Table 2-4: SPARQL Example Data [SPARQL]

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE
  { ?x foaf:name ?name .
    ?x foaf:mbox ?mbox }
```

Table 2-5: SPARQL: Example Query [SPARQL]

name	mbox
"Johnny Lee Outlaw"	<mailto:jlow@example.com>
"Peter Goodguy"	<mailto:peter@example.org>

Table 2-6: SPARQL: Example Query Result [SPARQL]

2.2.9 Web Ontology Language (OWL)

The Web Ontology Language (OWL) allows the definitions of ontologies which are formalized vocabularies of terms, often covering a specific domain and shared by a community of users. These definitions are specified by describing their relationships with other terms in the ontology. OWL can be divided in two specific subsets. OWL Lite for easy implementation and usage and OWL DL for the extensive support of Description Logic and to provide a language subset that has desirable computational properties for reasoning systems. The complete language is called OWL Full which relaxes some of the constraints of OWL DL so as to make available features which may be of use to many database and knowledge representation systems, but which violates constraints for Description Logic reasoning.

OWL itself is defined as extension of RDF/RDFS. This extension supports also the conformance to the RDF/RDFS given meaning by OWL. This results in a complete downward compatibility to RDF/RDFS.

Like RDF/RDFS OWL has several key elements described in detail below:

- **Classes** (owl:Class) allow the grouping of similar resources
    - **Class descriptions** are the basic building blocks for class axioms allowing the definition of classes
        - **Enumeration** (owl:oneOf) enables a class to be described by exhaustively enumerating its instances
        - **Property restrictions** (owl:Restriction) describe an anonymous class, namely a class of all individuals that satisfy the restriction.
        - **Intersection, union and complement** (owl:intersectionOf, owl:unionOf, owl:complementOf) represent the more advanced class constructors that are used in Description Logic. They represent the AND, OR and NOT operators on classes.
    - **Class axioms** allow the definition of classes utilizing class descriptions
        - **rdfs:subClassOf** allows one to say that the class extension of a class description is a subset of the class extension of another class description.
        - **owl:equivalentClass** allows one to say that a class description has exactly the same class extension as another class description.

        - **owl:disjointWith** allows one to say that the class extension of a class description has no members in common with the class extension of another class description.
- **Properties** define properties of an object (Object properties (owl:ObjectProperty) link individuals to individuals. Datatype properties (owl:DatatypeProperty) link individuals to data values.
    - **rdfs:subPropertyOf** (see RDF/RDFS)
    - **rdfs:domain** (see RDF/RDFS)
    - **rdfs:range** (see RDF/RDFS)
    - **owl:equivalentProperty** Expresses that two properties have the same property extension.
    - **owl:inverseOf** Defines the inverse relationship between two properties.
    - **owl:FunctionalProperty** A functional property is a property that can have only one (unique) value y for each instance x.
    - **owl:InverseFunctionalProperty** Asserts that a value y can only be the value of the property for a single instance x.
    - **owl:TransitiveProperty** Defines the property as transitive.
    - **owl:SymmetricProperty** A symmetric property is a property for which holds that if the pair (x,y) is an instance of P, then the pair (y,x) is also an instance of P.
- **Individuals** represent facts.
    - **owl:sameAs** indicates that an individual is the same as another individual.
    - **owl:differentFrom** indicates that an individual is different from another individual.
    - **owl:AllDifferent** indicates an set of different individuals.
- **DataTypes** representing data values
    - RDF Datatypes datatype scheme, which provides a mechanism for referring to XML Schema datatypes. Values are instances of rdfs:Literal.
    - **Enumerated datatype** using the owl:oneOf construct.

Further constructs are annotations, ontology header, imports and version information.

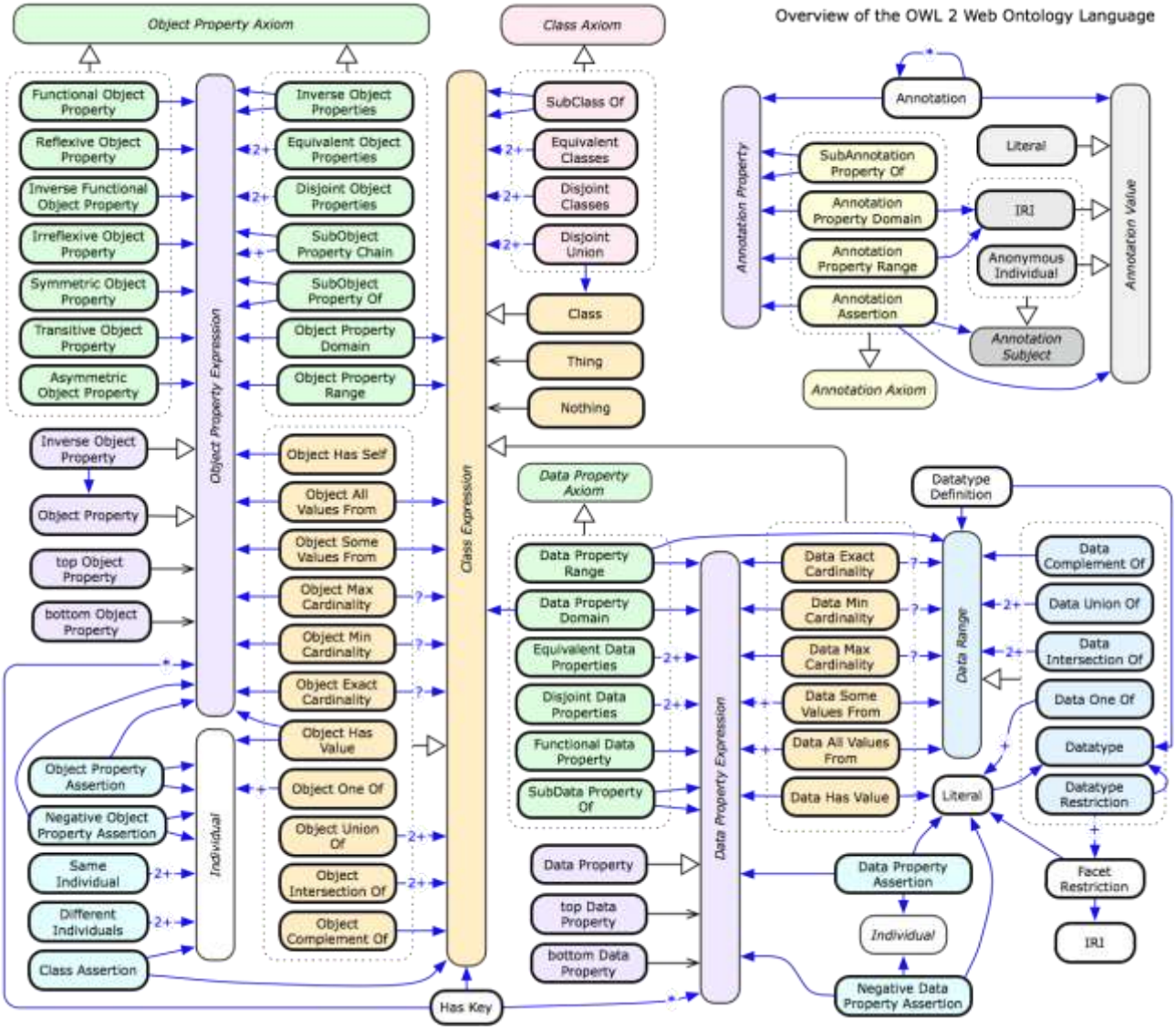

Figure 2-11: OWL 2 Overview

2.2.10 Relation an explanation MOF, UML, OCL, QVT and RDF, RDFS, OWL

In this section an overview of the relation between the OMG and W3C technologies is given. It's hard to compare the technological approaches. However, it is clear that RDF based technologies are more fine-grained than the MOF based technologies. The smallest information containing unit for RDF are simple triples whereas for MOF one has to deal with heavy objects containing slots with values adhering to the structure of a given meta-class.

Most concepts in RDF are first class entities, meaning that they can be used without another entity. This is not the case in MOF where a property has to be contained in a class a cannot exist otherwise. This comes through the strong containment relationship (aligned to document definitions in XML) introduced in MOF2. There is no direct support for this kind of relationship in RDF which makes it more flexible but limits also consistency functionality within the object lifecycle.

All this and more makes a comparison hard. The following table tries to compare the technologies in a simple fashion, indication possible differences and problems in projection between the technologies. In rows with all same values it is less likely to have mapping problems. Functionality which is only supported by one technological framework is likely to cause problems in latter mapping implementation, since it might not be easy to preserve the information and semantics behind.

	MOF	MOF, OCL	MOF, OCL, QVT	RDF, RDFS	RDF, RDFS, OWL
Class / Classifier					
Independence	Yes[2]	Yes[2]	Yes[2]	Yes	Yes
Derivation	No	No	Yes[3]	No?	Yes
Disjoined	Yes[4]	Yes[4]	Yes[4]	No	Yes
Final	Yes[5]	Yes[5]	Yes[5]	No	Yes
Ordered	No	No	No	No	No
Unique	No	No	No	No	No
Multiplicity	No	No	No	No	Yes
Property					
Independence	No[6]	No[6]	No[6]	Yes	Yes
Derivation	Yes[7]	Yes	Yes	No	No
Disjoined	No	Yes[8]	Yes[8]	No	No
Final	No	No	No	No	No
Ordered	Yes	Yes	Yes	No	Yes
Unique	Yes	Yes	Yes	No	Yes
Multiplicity	Yes[9]	Yes[9,10]	Yes[9,10]	No	Yes
Relation					
Independence	No	No	No	Yes	Yes
Derivation	Yes	Yes	Yes	No	Yes
Disjoined	Yes	Yes	Yes	No	No
Final	No	No	No	No	No
Ordered	Yes	Yes	Yes	No	Yes
Unique	Yes	Yes	Yes	No	Yes
Multiplicity	Yes[9]	Yes[9,10]	Yes[9,10]	No	Yes

Table 2-7: Relation MOF, OCL, QVT and RDF, RDFS, OWL

[2] With exception do the model which is the general container and therefor neglectable.
[3] Via QVT transformation rules (e.g. relations of QVT Relations)
[4] Via the inheritance relation
[5] If marked as leaf.
[6] Depends on Class/Classifier
[7] It can be marked as derived. Full functionality comes from support for expressions and constraints.
[8] There is no direct support but via constraints.
[9] Support for upper and lower bound via constant numbers with support for infinity.
[10] Advanced multiplicity constraints (e.g. even amount, twice as another property, …)

2.3 Transformation languages

Model transformation in MDE can be categorized into model to model (M2M), text to model (T2M), and model to text (M2T). Since T2M and M2T concentrate on structural mediation the focus of the document is on M2M.This section deals with model-to-model transformations. After a brief introduction to categorization on model transformation QVT and ATL are presented completed by a comparison.

2.3.1 Categorization

This categorization is based on the work in [Czarnecki, 2003] and referred to in many publications (such as [Jouault, 2006] or [Kurtev, 2006]). In this section, the features of model transformation languages will be put forward.

2.3.1.1 Transformation rules

In model transformation languages, the transformations will be described by the means of rules, which relate elements of the source and the target model. These rules consist of a left-hand side (LHS) that specifies which elements of the source model are to be included in the transformation, as well as a right-hand side (RHS), that hence relates to the elements of the target model. The elements that the rules are composed of are:

- Variables are used to hold elements from the source or target model. To help distinguish them from variables in the model (which may represent a Java class with variables), they are also referred to as meta-variables.
- Patterns are fragments of the model and may contain an arbitrary number of variables. The can be distinguished into string, term and graph patterns.
- Logic is used to express constraints and actions on the model elements. Logic can be further divided into non-executable logic, which is used to describe relations between model elements, and executable logic. Executable logic differentiates between a declarative and an imperative approach.

Declarative logic expresses rules by defining pre- and post-conditions that must hold for both the source- and the target model. The pre-condition describes the state of the system before the transformation and the post-condition accordingly the state after the transformation. As this approach allows more abstract transformation rules, it is suited for abstract or incomplete transformations. It is used by graph transformations or e.g. XSLT.

Imperative logic describes the transformations with the help of operations. These often can be executed directly, which results in more efficient implementations — declarative logic expressions are abstract and have to be converted into actual transformation code before they can be executed. But often the semantics of imperative rules are not equally powerful in describing complex situations, like difficult object pattern matching operations. They tend to be more verbose, whereas they can be simpler to use when populating model elements. This shows that an ideal transformation language should provide both declarative and imperative approaches, so that the appropriate solution can be chosen according to the specific problem domain.

Another important feature of transformation languages is the kind of directionalities it supports. Directionality describes in which directions the transformations can be executed. Two approaches are identified by: a one-way transformation can only be executed in one direction, whereas two-way transformations may be executed for- or backward. This has a great impact on the power of the language, since in a unidirectional language one needs to define two separate transformations to have the same functionality that one can achieve in a bidirectional language with just one transformation.

2.3.1.2 Source-Target relationship

This category differentiates between the kind of relationships that exist between source and target models of the transformation. It states two possibilities: the basic one where the source and target models are different

and one where the source- and target models are the same. This is referred to as “in-place” updating of the model. This may introduce more complexity in the transformation process — e.g. when a transformation creates an element that is then again matched by the transformation rules and creates another element that in turn is matched again and so forth.

2.3.1.3 Rule Application Strategy

Transformations are executed by finding matches for pattern in the source model. Since it is possible that a pattern matches more than one element, a strategy must be defined that states in which order the rules will be executed on the matches. In three possibilities are given:

- Deterministic — where a algorithm determines the exact order of rule execution.
- Non-deterministic — where multiple transformation executions may yield different rule application orders.
- Interactive — were the user has to define the order of execution of the rules.

2.3.1.4 Rule Scheduling

The rule scheduling determines in which particular order the rules of a transformation are executed. One of the concerns is the form in which the order is expressed. This may be:

- Implicit scheduling depends on relationships between rules that are not directly expressed in the rules itself
- Internal explicit scheduling relies on the explicit appointment of the schedule through the use of control flow structures or triggering of other rules in the rule definition itself, whereas
- External explicit scheduling uses scheduling logic that is separated from the rule logic itself

Other areas that can be differentiated are the rule selection which deals with rule conflict resolution when multiple rules apply to the same elements, rule iteration that defines how rules can be executed multiple times and phasing, where rules are assigned to certain phases in the transformation.

2.3.1.5 Rule Organization

This category deals with the mechanisms a transformation language provides to organize the rules. Since complex modeling environments may produce a great amount of transformation rules, it is needed to provide a way for the arrangement or subdivision of the rules. The possibilities are:

- Modularity mechanisms allow the organization of rules into modules or libraries. These can import other modules and be imported themselves by other transformation definitions.
- Reuse mechanisms are those that allow the definition of rules based on other rules. This can be implemented through the use of rule inheritance, derivation, extension and specialization. Similar techniques may be applied to modules, e.g. module inheritance.
- Organizational structure where the rules are laid out similar to the source languages structure. For example, in a language like UML where Packages can contain Classes and these can contain Attributes, there may exists rules for the creation of Packages and these rules may contain rules for the creation of Classes. These rules in turn may contain the rules for the creation of Attributes and so forth.

2.3.1.6 Traceability Links

Traceability Links are used to connect the source- and target elements that are created during the course of the transformation. These links are later be used for, e.g. change impact analysis or requirements validation. The transformation language may support the automatic creation of traces during the transformation execution, allow for the manual adding of these relationships and some may not provide any traceability support at all. In this case, traceability links may be added to the model just like any other model element. When the traces are created automatically, it may also be differentiated between the places where these links are stored — either inside the source or target-models or separately.

After this introduction into model transformation, model transformation languages and an approach for their categorization, two model transformation languages will be introduced: the Query/View/Transformation and

the Atlas Transformation Language. They will then be compared according to the categorization given in the last section.

2.3.2 QVT

Is a model-based transformation language that allows specifying transformations between different models based on their metamodels. QVT Relations provides both, a textual and graphical syntax. Figure 2-12: QVT: UML Class to Relational Table Relation - Graphical Notation shows the “UML Class to Relational Table” relation in graphical syntax and Figure 2-13: QVT: UML Class to Relational Table Relation - Textual Notation illustrates its textual pendant. Traces are implicitly created. A trace model defines relationships between source and target model elements, i.e. an instance of a trace class specifies which element in the source model a target model construct originates from. This information is used to realize incremental updates to a target model to avoid re-executing of entire transformations. QVT Relations allows the definition of complex object patterns for matching elements in a model.

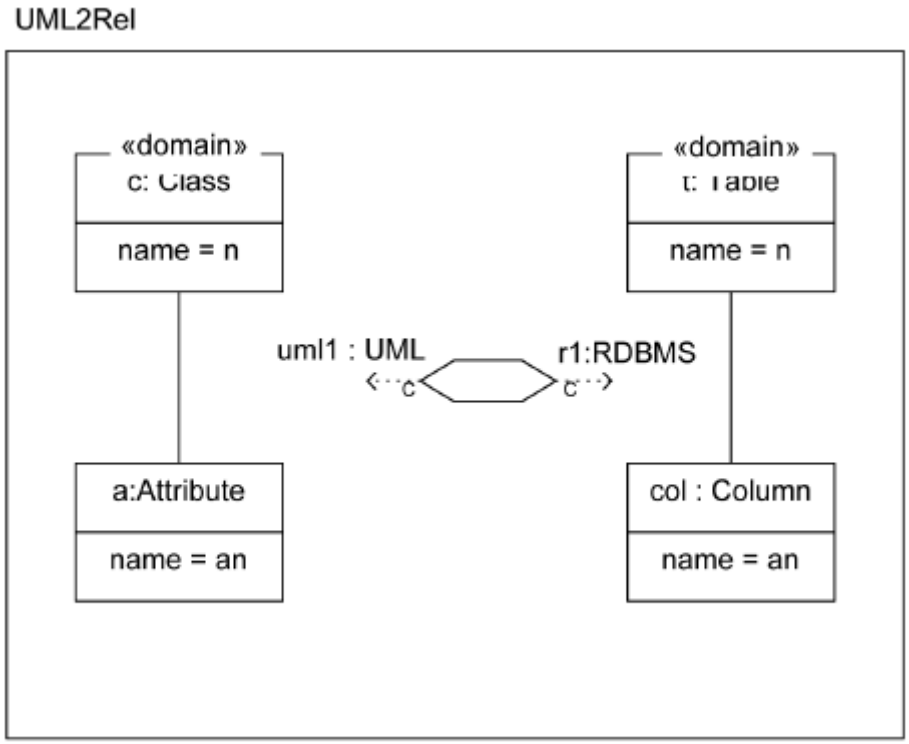

Figure 2-12: QVT: UML Class to Relational Table Relation - Graphical Notation

QVT offers also an imperative approach to define transformations. It can either be used to define complete transformations or to be called from a Relations relation to express parts which are difficult to describe in a declarative manner. The QVT Operational Mapping syntax is similar to nowadays imperative programming languages. As can be seen in Figure 2-8: QVT Package Overview, it is related to the QVTRelation package, because it uses the same traces framework. Otherwise QVT Operational Mapping is completely independent from the other parts as it defines its own imperative extensions to the EssentialOCL constructs.

```
1  relation UML2Rel {
2    checkonly domain uml1 c:Class {
3      name = n,
4      attribute = a:Attribute{
5        name = an
6      }
7    }
8    checkonly domain r1 t:Table {
9      name = n,
10     column = col:Column{
11       name = an
12     }
13   }
14 }
```

Figure 2-13: QVT: UML Class to Relational Table Relation - Textual Notation

Furthermore, QVT allows the definition of so called Black-Box implementations to specify a transformation in arbitrary programming languages. This is meant to use domain specific libraries for transformations or for legacy support [Wagner 2005]. For better understanding, the QVT standard states an analogy between the different parts of QVT and the Java Virtual Machine. Table 2-8: QVT: Java analogy shows the analogous elements.

Java Element	QVT Pendant
Java Byte Code	Core language
Behavior specification of Java Virtual Machine	Core semantics
Java language	Relations language
Java compiler specification	Relations to Core Transformation
Java Native Interface	Imperative parts

Table 2-8: QVT: Java analogy

Figure 2-14: QVT: Meta-model Relationships summarizes the relationships between the QVT parts. It depicts the transformation from Relations to Core. The Operational Mappings as well as Black-Box implementations depend on the declarative parts for implementing trace handling.

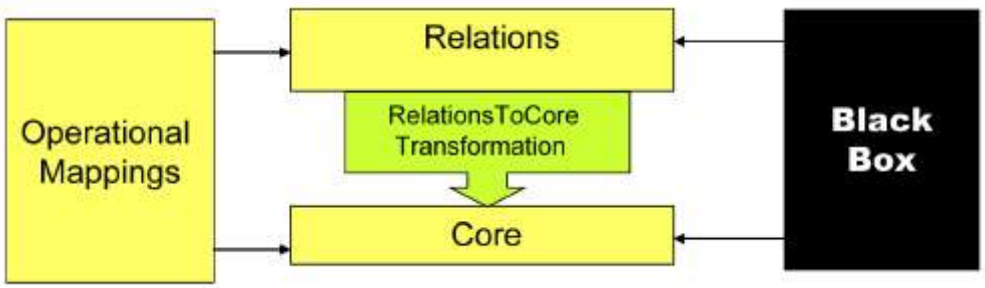

Figure 2-14: QVT: Meta-model Relationships

2.3.2.1 Concepts of the relational transformations

In the QVT relations language, a transformation is nothing but a set of relations. To be considered a successful transformation, all relations that are contained within it have to hold. A relation is a description of

constraints which have to be satisfied by the elements of a model. A transformation consists of one or more models. These models have to be instances of meta-models, which in turn need to be instances of the MOF meta-model. For every transformation execution, the direction in which the transformation will be performed must be given. This is done by specifying the target model, which may be empty or not. The execution itself is fulfilled by checking if all the relations of the transformation hold. If they do not hold, the target model is altered to enforce the relationships. The modifications that can be applied to the target model include the creation, deleting or changing of model elements. A relation can be marked as top. All top relations of a transformation are checked if they hold, if they are not marked as top they will only be checked if they are referenced by another relation.

The model elements that are to be associated in a relation are determined by a feature called domain. A domain is a subset of the model that is, in any way, distinguishable from other elements of the model. It consists of pattern, which can be seen as object templates. The patterns describe elements and certain attributes or associations of it. This pattern is then matched against the elements contained in the model. If the pattern of a domain of the transformation target model does not match to any element of the model, there are two possible actions: if the domain is marked as checkonly nothing will be done. No elements will be added or changed. If the domain is marked as enforce, then the steps necessary to make the relationship hold will be executed.

Besides the domains and their patterns, relationships also consist of variable declarations, when and where-clauses. Variables can be used to set fields of different domains in relation. They have a name and a type and are declared at the beginning of the transformation definition. The when-clause of the relation is used to specify additional constraints that must hold. It can include OCL constraints or reference other relations. The where-clause on the other hand is used to specify additional constraints that must hold only if the relationship itself holds. This can be used to add properties or associations to elements that where created in the relation before.

To get a better understanding of the syntax of the QVT relations language, the next section will give an example transformation between two basic meta-models.

2.3.3 ATL

The Atlas Transformation language (ATL) [ATLAS-ATL] is a model transformation language. It is being developed as part of the Atlas Model Management Architecture (AMMA) [ATLAS-AMMA] at the University of Nantes. It was issued as a proposal for the OMG MOF/QVT RFP. ATL is a hybrid language; it incorporates both declarative and imperative constructs. Although its creators encourage the use of the declarative language, it is recognized that some transformations can be better specified in a imperative way. ATL has an abstract syntax which is described as an MOF meta-model, a concrete textual syntax and a graphical representation. The ATL works on models that are instances of a MOF compliant meta-model. Note that this is the same approach that the QVT implementation has followed.

Transformations in the ATL are unidirectional, that means they can only be declared and executed in one direction. However, bidirectional transformations can be implemented by specifying two distinct transformations, one for each direction with opposite semantics. ATL operates on source- and target models. The source models are read-only. This means that it is only possible to navigate through these model but no changes can be applied to them. The target models in contrast are write-only. It is not possible to navigate through the elements of these models but elements can be created or altered. Transformations in ATL are organized in modules, which in turn consist of a header, imports, helpers and transformation rules.

- In the header, the name of the transformation module is declared as well as the participating model elements and their meta-models. There need to be at least one in- and one output-model, but more are possible.
- In the import section, additional ATL libraries can be imported into the module. These libraries may be both provided and standardized by the ATL engine or self-defined libraries that include often used helper methods.

- The helper section is the place to define chunks of code that can be accessed from the transformation rules. These so-called helpers are like procedures that can be invoked. They have a name, a set of parameters, a return type and a body. Note that it is not possible to alter the target model in a helper method; they are only used to extract common functionality.
- In the rules section, the transformation rules themselves are declared. The next section will give an overview over the kinds of rules and their definition.

The next section will deal with the most important aspect of a model transformation language, the definition and execution of transformation rules.

2.3.3.1 Transformation rules

As stated before, the ATL consists of a declarative and an imperative approach for defining transformation rules. Because section 2.3.2.1 Concepts of the relational transformations about QVT had the focus on the declarative part, this section about ATL will also concentrate on this approach.

Declarative Rules in ATL are also called matched rules. They are made up of a source and a target pattern. The source pattern is describing elements that will be searched for in the source model. It consists of multiple source types — including types from the source meta-model or standard OCL collection types — and a guard, which is a Boolean typed OCL expression. When the transformation is executed, the source pattern is evaluated and a set of the matching elements is created. The target pattern is made up of multiple elements, each defining a target type that has been defined in the target meta-model and bindings. The bindings are used to initialize the features of the target elements, like properties or associations. The listing Figure 2-15: ATL transformation rules (matched rules) shows a basic transformation. The source pattern can be found on line 2. It consists of a type Class from the MyUmlMM meta-model. The guard is the expression in parenthesis. This pattern will match all instances of the type Class in the model which have a name that is not equal to the empty String. The target pattern is found on line 5 and consists of the target type Table from the MyDbMM meta-model. The instances of this type that are created in the course of the transformation will have their table name attribute initialized with the name value of the class from the source pattern. In the declarative rule, there may be an imperative part after the target pattern. It is started by the do keyword and may include imperative code such as loops etc. to enrich or further initialize the target model. It is this mixture of declarative and imperative rules that make the ATL a hybrid transformation approach.

```
1  rule Class2Table {
2    from
3      c : MyUmlMM!Class (not (c.name = ''))
4    to
5      t : MyDbMM!Table (
6          table_name <- c.name,
7          table_columns <- c.attributes
8      )
9  }
10
11 rule Attribute2Column {
12   from
13     a : MyUmlMM!Attribute
14   to
15     cn: MyDbMM!Column (
16         column_name <- a.name,
17         column_type <- a.type
18     )
19 }
```

Figure 2-15: ATL transformation rules (matched rules)

Similar to the top relations of QVT, the ATL provides a feature to restrict the execution of rules. There are standard rules that have behaviour similar to top relations in QVT, meaning they are executed for every match of the source pattern. Then there are lazy and unique lazy rules which are only executed when they are triggered from other rules. The difference between the lazy and unique lazy rules is that the latter, when called several times, will re-use the elements it had created when it was triggered before. The lazy rules will generate all elements from the target model again if it is called multiple times.

Another similarity to the QVT is the automatic creation of traceability links between elements of the source- and the target model that have been created in the course of the transformation.

Imperative Rules Imperative rules are referred to as Called rules in ATL. Like helpers, called rules have to be triggered by other rules to be executed and they also accept parameters. But helpers are not allowed to alter the target model whereas called rules can. The triggering of called rules is restricted to those code sections that are themselves imperative, either in a matched or called rule.

The main difference to matched rules is not only the imperative nature but the fact that since it is called, it does not have to match an element in the source model. Therefore, the from part of matched rules is not allowed in called rules. It consists only of a target pattern and an optional imperative do section.

Its main purpose though is to generate elements in the target model and initialize them, without these elements having a match in the source model.

Execution modes ATL transformations can be executed in different modes. This separation is done in order to allow more basic transformation definitions when possible.

In the normal execution mode — ATLs default mode — the transformation is executed as described in the previous section. By executing the rules (matched or called), the elements of the source model that are to be transformed into the target model are found and the appropriate elements in the target model are created. This mode should suit the most use-cases, as it allows for the transformation into a totally different target model. These kinds of transformation are called vertical transformations. The second execution mode is called the refining mode. It is intended for those transformations that are executed only to enhance some elements of one model instead of creating a totally different one. It is enabled by specifying the keyword refines instead of from in the header of the ATL transformation file. The definition of these refinements is done via matched rules. However, it is only needed to specify the elements that are to be refined and the changes that have to be executed on them. All other elements that do not change are simply copied by the ATL engine into the target model.

After this overview of the ATL transformation language, the QVT and ATL languages will be compared according to some of the features of model transformation languages that were introduced in section 4.1.

2.3.4 Comparison of QVT and ATL

In this section, the features of the QVT and ATL transformation languages will be compared and the reason why the QVT was chosen as the model transformation language for this thesis. The comparison will use the classification form section 2.3.1 to find differences or similarities in the language definitions.

2.3.4.1 Transformation rules

Both QVT and ATL allow the definition of model transformation rules with the help of variables, patterns and logic. The QVT has two distinct approaches: the declarative part consisting of the Relations and Core language, and two imperative approaches: the Operational Mappings and the Black-Box Operations. The ATL on the other hand is a hybrid language that incorporates both declarative and imperative approaches.

Both transformation languages allow the definition of model patterns using a textual or graphical notation, the abstract syntax of both is described as a MOF meta-model and they are using the meta-model of the source- and target-model to define the types of the elements used in the transformation.

In the QVT, the LHS or RHS of the transformation are marked indirectly as the domains of the rules are connected to models — which in turn are used to set the direction of the transformation. So by stating that domainA belongs to modelA and that the transformation will be executed from modelA to modelB, the domainA can be identified as the LHS. In the ATL, the LHS is stated explicitly by the keyword from and the RHS is marked with the keyword out in the transformation rule.

2.3.4.2 Relationship between Source and Target

In the QVT, an arbitrary number of models can be used in a transformation. An in-place model transformation is possible through the binding of the source- and target model to the same model instance. The ATL allows only for separate source-and target models. In-place updates are not possible, with the exception of the refinement feature of a transformation.

2.3.4.3 Rule scheduling

As stated, scheduling mechanisms “determine the order in which individual rules are applied”. The QVT only has two different types of rule scheduling: “normal” and top relations. Top relations are always checked for conformance, whereas non-top rules are only checked when they are triggered from other relations. In accordance to the categorization given by, the top relations are of an implicit nature and the non-top rules are of explicit form. ATL has similar concepts. Matched rules have an implicit form, since they are called by the engine whenever their source pattern matches in the appropriate model. Called rules however are of an explicit nature, since they have to be called explicitly to be executed. So both languages support implicit and internal explicit forms of rules scheduling.

2.3.4.4 Traceability Links

In the QVT relations language and in the operational mappings, the trace links between the source and target elements of a transformation are created automatically. In the core language, these links have to be established by hand. The place where these links are stored is only determined for the core language. The core stores the links in a separate model, whereas the QVT specification makes no statement about the other languages. It is therefore up to the particular QVT engine implementation to decide where to store these links.

In ATL, the tractability links are created automatically, too. The storage of these links is handled by the transformation engine. The ATL specification makes no statements about where they have to be stored.

2.3.4.5 Rule Organization

In the QVT relations, there exists no support for modules or relational inheritance. In the operational mappings however, the definition and import of modules is possible, as well as the extension of transformations. The language provides concepts like libraries, helpers, mappings merge and mappings inheritance.

The ATL allows the definition and importing of modules and libraries, too. ATL also provides mechanisms for rule inheritance.

2.3.4.6 Directionality

The QVT relations language allows bidirectional transformations. The transformations and rules only use elements or domains that belong to a certain model. The direction of the transformation is then specified when triggering it. In the ATL, transformations are only unidirectional as the in- and output models are fixed and it is impossible to navigate the target- or change the source-model.

2.4 Information integration and semantic mediation approaches

In this section different approaches for information integration and semantic mediation are presented.

2.4.1 Common Meta-Model Approach

The common metamodel approach as proposed for example in the EU Project Speeds [Speeds] and in the ARTEMIS-JU Project CESAR [CESAR] utilizes the same metamodel for the description of all information. Since for legacy and 3rd Party tools this way of storing information is not native, adaptation is necessary. In this case semantic mediation is implemented in the adapters where information is first extracted, semantically transformed, and then stored in the central storage and representation format (see: Figure 2-16: Common Meta-Model Approach).

This approach has several advantages:

- This approach is **easy to understand** because it is, from an architectural viewpoint, simplistic. This is not to be under estimated, since this is the first challenge in the applicability of an approach.
- It is relatively **easy to be implemented** since the necessary technology is available and well proven. Main challenge in this approach is in the modeling in the data representation itself (e.g. metamodel creation). The metamodel has to support the representation of all the necessary tool information in on common structure.
- Since this approach has to support only one metamodel storage (e.g. Repository) for this metamodel can be **better optimized with regard to performance and consumed space** in contrast to more generic storages. Because of the more concrete abstraction level more concrete operations can be provided and understood by the tools and tool adaptors.

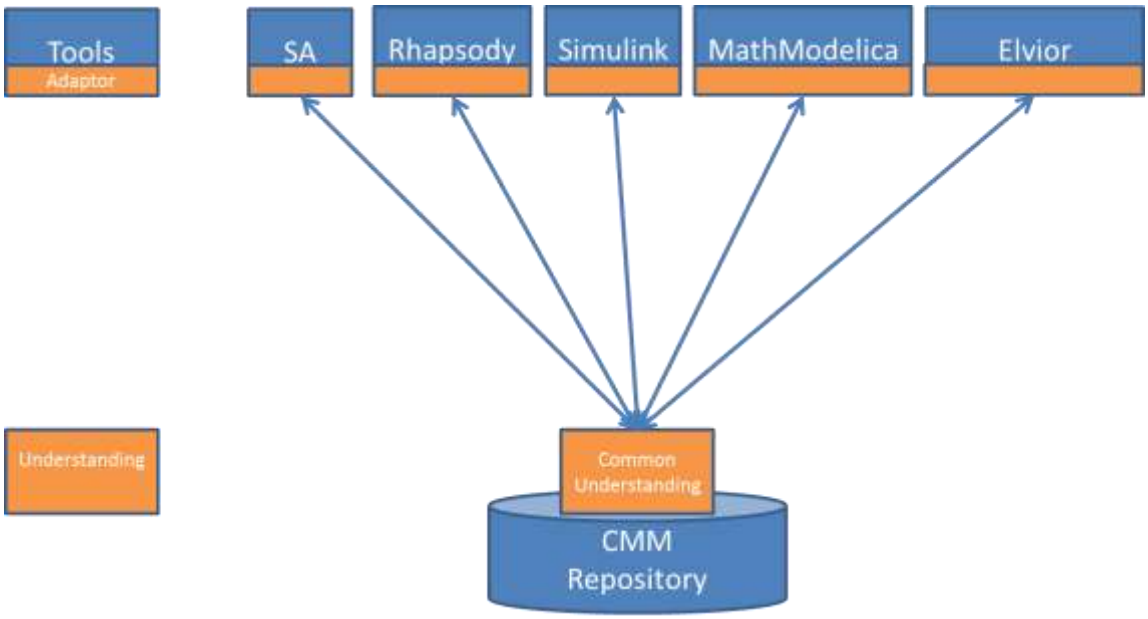

Figure 2-16: Common Meta-Model Approach

However, this approach has also several disadvantages:

- It has a problem with regard to **architectural scalability**. As one can see separation of concern only happens on the level of the metamodel. The approach does not allow for introduction of custom viewpoints and abstractions. Changes have to be agreed by all involved parties.
- The approach has also drawbacks with regard to **extensibility, enhancements and flexibility**. The addition of a tool to this approach can have severe impact if the tools information representation requires a nonconforming or extensive change to the common metamodel, in particular if new concepts are conflicting with existing ones. The worst possible impact is that all existing tool

adapters have to be changed. The change might have to be implemented in different languages depending on the diversity of the tool adapter implementations.

- The **opaque realization of the semantic mediation** in the tool adapters represents **hidden and implicit information** not available to the other tools which might.

The approach is also in conflict to the Domain Specific Language (DSL) paradigm and the experience with the UML specification (one language for all) whose failure leads now to the simplification of UML.

Why standardization of information representation and structure?

This kind of standardization is fostered to achieve tool interoperability and foster data accessibility. It is also used to exchange data between companies and or departments within companies. However, the approach of standardizing the whole word does not seem to be very successful at this time and should better by avoided.

2.4.2 Transformation Chain Approach

The transformation chain approach as for example proposed in ModelBus [ModelBus] and in the ARTEMIS-JU Project iFEST [iFEST] utilizes a multitude of metamodels for the storage of all information and model transformations for semantic mediation. This approach allows legacy and 3rd Party tools to use a metamodel native to them without concern about semantic mediation. In this case semantic mediation is usually implemented outside adapters where information is only extracted, and then stored in the central storage (see Figure 2-17: Transformation Chain Approach). Semantic mediation is applied via model transformation. This can be collocated on the repository server or on other servers invoked remotely. These invocations can be embedded in a development process deployed to a process engine observing events or directly embedded i a server logic ensuring that the established traces stay up to date.

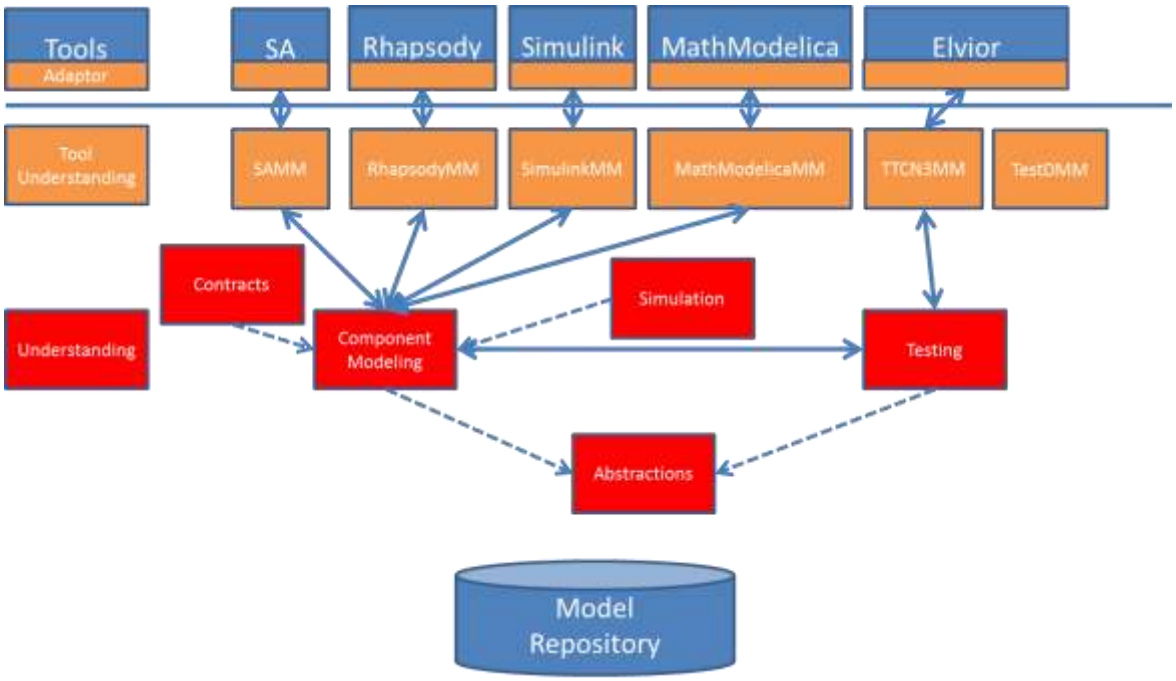

Figure 2-17: Transformation Chain Approach

This approach has several advantages:

- The approach is **flexible and extendable** since new metamodels and mediating transformations can be added. New abstractions can be introduced in a flexible way.
- **Scaling and performance** with regard to read operations due to materialized views created through model transformations.
- **Reliability and technological robustness** through years of technological availability and experience.

And disadvantages

- The approach faces **synchronization overhead** through computation of model transformations in order to keep the information in sync and consistent.
- There is also some overhead with regard to the **management** of the models, metamodels and transformations (e.g. when to transform, consistency between transformation, cyclic dependencies)
- The storage of **redundant information** has also an impact on the amount of needed **storage space**.
- The approach **poses challenging requirements** on the functionality of transformation languages and engines

2.4.3 Sea of Information Approach (Ontologies)

The sea of information approach is new to the engineering domain and has not been applied in this area as to the public knowledge. The approach fosters the use of semantic web technology for the establishment of a flexible and fine grained extendable common understanding. Because of this focus it is not to be understood as a replacement for specific transformation like test generation.

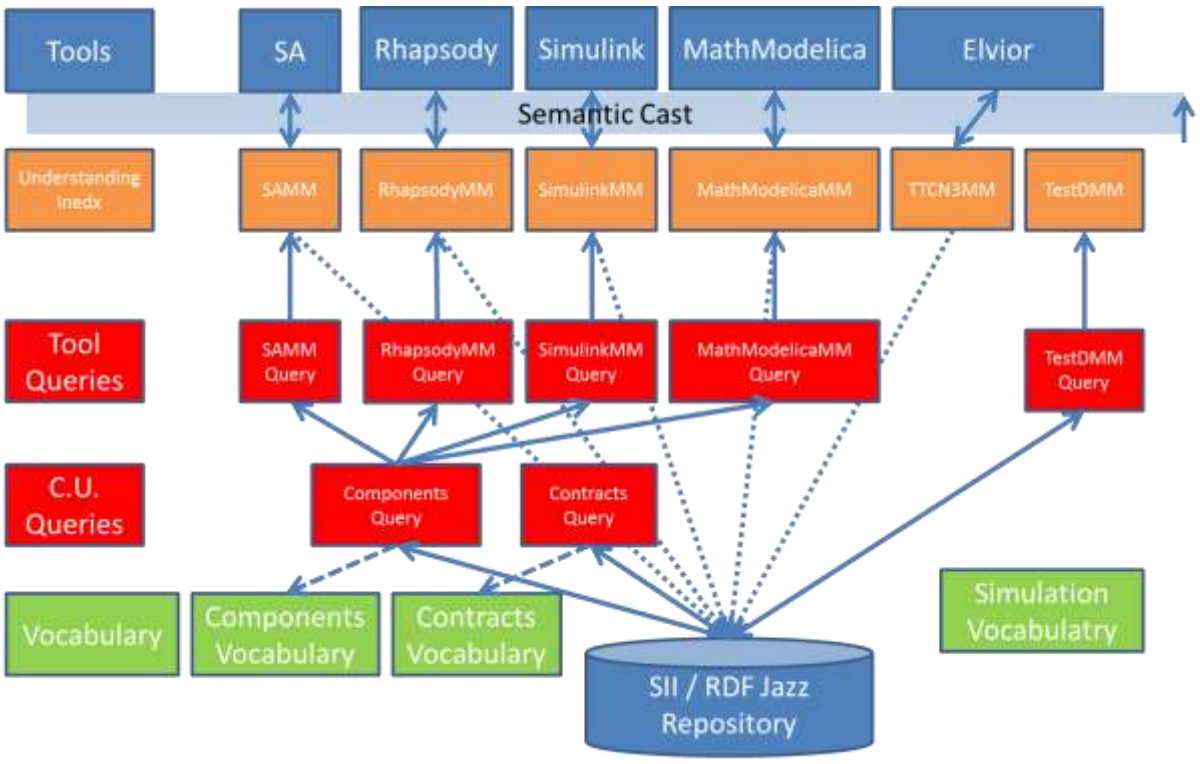

Figure 2-18: Sea of Information Approach

This approach has several advantages:

- Because of the technologies the approach is has an high degree in **openness, flexibility and extensibility**
- **Redundancy** can be limited to a minimum (e.g. every object exists (ideally) once and can have all its properties)

And disadvantages

- The approach utilizes **technologies new** to the engineering domain hence with unknown characteristics with regard to **technology, abilities and scalability**
- More **overhead for acquiring the knowledge** to handle the undying technology (especially since the approach opens so many possibilities)

3 Relevant Concept Spaces for the SPRINT Semantic

This section lists concept spaces of relevance for Sprint. The list will evolve during project time.

3.1 Heterogeneous Rich Component (HRC)

Category	Information
Originated from	SPEED EU Project
Abstraction Level and Domain	Systems Engineering Domain Component Oriented Design Contract-Based Methodology Behavioural Modelling State Machines
General information	The HRC meta-model is defined to be the language to represent systems in terms of rich component models. This meta-model must be, on one hand, expressive enough to cover all the features required by the HRC process and, on the other hand, easy to understand and convenient to use both by HRC users and analysis tools. As a result the HRC meta-model consists of three levels.
Originated from	
Resources	http://www.speeds.eu.com/index.php?option=com_content&task=view&id=23&Itemid=41

3.2 Cesar Common Meta Model (CMM)

Category	Information
Originated from	CESAR ARTEMIS-JU Project
Abstraction Level and Domain	Systems Engineering Domain Component Oriented Design Behavioural Modelling State Machines Variability Modeling
General information	See HRC
Originated from	SPEED EU Project
Resources	http://www.fokus.fraunhofer.de/en/fokus_events/motion/mdtpi_2010/_docs/MDTPI04.pdf

3.3 Open Services for Lifecycle Collaboration (OSLC)

Category	Information
Originated from	IBM
Abstraction Level and Domain	Domain Independent on a meta level

Category	Information
General information	Open Services for Lifecycle Collaboration (also known as OSLC or Open Services) is an open community dedicated to breaking down the barriers between the tools in the product and application lifecycle by making it easier to use lifecycle tools in combination.
Originated from	
Resources	http://www.fokus.fraunhofer.de/en/fokus_events/motion/mdtpi_2010/_docs/MDTPI04.pdf

3.4 AUTOSAR

Category	Information
Originated from	AUTOSAR.ORG
Abstraction Level and Domain	Automotive Domain Variability Modelling Component Modelling Systems Engineering
General information	AUTOSAR (AUTomotive Open System ARchitecture) is an open and standardized automotive software architecture, jointly developed by automobile manufacturers, suppliers and tool developers. • paves the way for innovative electronic systems that further improve performance, safety and environmental friendliness • is a strong global partnership that creates one common standard: "Cooperate on standards, compete on implementation" • is a key enabling technology to manage the growing electrics/electronics complexity. It aims to be prepared for the upcoming technologies and to improve cost-efficiency without making any compromise with respect to quality • facilitates the exchange and update of software and hardware over the service life of the vehicle
Originated from	
Resources	http://www.autosar.org/

3.5 EAST-ADL

Category	Information
Originated from	ATESST EU-Project
Abstraction Level and Domain	Automotive Domain Variability Modelling Component Modelling

Category	Information
	Requirements Engineering Safety Behavioural Modelling Timing
General information	EAST-ADL is a modeling language defined as a domain-specific language for the development of automotive electronic systems. The modeling element consists of different entities to represent the embedded system (requirements, features, desired behaviors, software, and hardware components) and their dependencies (refinement, allocation, composition, communication, etc.).
Originated from	
Resources	www.atesst.org

3.6 Architecture Analysis and Design Language (AADL)

Category	Information
Originated from	sae.org
Abstraction Level and Domain	Behavioural Modelling Runtime Analysis Timing Systems Modelling
General information	This standard defines a language for describing both the software architecture and the execution platform architectures of performance-critical, embedded, real-time systems; the language is known as the SAE Architecture Analysis & Design Language (AADL). An AADL model describes a system as a hierarchy of components with their interfaces and their interconnections. Properties are associated to these constructions. AADL components fall into two major categories: those that represent the physical hardware and those representing the application software. The former is typified by processors, buses, memory, and devices, the latter by application software functions, data, threads, and processes. The model describes how these components interact and are integrated to form complete systems. It describes both functional interfaces and aspects critical for performance of individual components and assemblies of components. The changes to the runtime architecture are modeled as operational modes and mode transitions.
Originated from	Avionics Architecture Description Language
Resources	http://www.aadl.info

3.7 Unified Profile for DoDAF/MODAF (UPDM)

Category	Information
Originated from	OMG
Abstraction Level and Domain	Capabilities Integration and Development Planning, Programming, Budgeting, and Execution Acquisition System Systems Engineering Operations Planning Capabilities Portfolio Management
General information	UPDM is an Object Management Group (OMG) initiative to develop a modeling standard that supports both the USA Department of Defense Architecture Framework (DoDAF) and the UK Ministry of Defence Architecture Framework (MODAF). The modeling standard is called the Unified Profile for DoDAF and MODAF (UPDM).
Originated from	Department of Defense Architecture Framework (DoDAF) British Ministry of Defence Architecture Framework (MODAF)
Resources	http://www.aadl.info

3.8 UML Testing Profile (U2TP)

Category	Information
Originated from	OMG
Abstraction Level and Domain	Testing
General information	The UTP provides extensions to UML to support the design, visualization, specification, analysis, construction, and documentation of the artifacts involved in testing. It is independent of implementation languages and technologies, and can be applied in a variety of domains of development.
Originated from	
Resources	http://utp.omg.org/

3.9 Test and Performance Tools Platform (TPTP)

Category	Information
Originated from	ECLIPSE
Abstraction Level and Domain	Testing
General information	TPTP addresses the entire test and performance life cycle, from early testing to production application monitoring, including test editing and execution, monitoring, tracing and profiling, and log analysis capabilities. The platform supports a broad spectrum of computing systems including embedded, standalone,

Category	Information
	enterprise, and high-performance and will continue to expand support to encompass the widest possible range of systems.
Originated from	
Resources	www.eclipse.org/tptp/

4 Technological recommendations

In this section recommendations for further development are discussed.

4.1 Modelling framework

For the use of modelling framework RDF(S) and OWL as well as possible extensions are recommended. Because:

- Since multiple classification is allowed, data can more easily be reused without redundancies
- They allows a higher degree of abstraction than for example MOF (triplets represent a very abstract and simplistic from of data representation)
- They are more fine grained and therefore more universal
- They have better support for OWA, which is necessary in the addressed setting
- They are near to the internet and evolving internet technologies (W3C)

On the downside, RDF-related technologies provide a very low level programming model and it becomes an overhead on the overall development phase of the eventual adapters of the tools to be interfaced with the framework. For this purpose, we shall consider the EMF technology as an entry-point towards the lower level RDF programming model. Lack of support for this technology is seen as a mayor risk for the acceptance of the project results.

4.2 Information integration and semantic mediation

The sea of information approach scenario because more flexible, extensible and reflects more the OWA. Key points for this recommendation are:

- The extensibility of the approach
- Good support through the recommended modelling framework (Multi viewpoint support down to object level)
- Good coverage for most of the needed semantic mediation needs with simple and available mechanisms
- Consistency support

However, this approach can be enhanced with legacy functionality form the transformation chain approach. Legacy functionality support is especially necessary to support existing model transformations or to support deeper semantic mediation with very intricate rules.

5 Abbreviations and Definitions

Architecture	The fundamental organization of a system (maybe on different abstraction levels) embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.
Artefact	A physical piece of information that is used or produced by a software development process. Examples of Artefacts include models, source files, scripts, and binary executable files.
ATL	ATL (ATL Transformation Language) is a model transformation language and toolkit. (See: http://eclipse.org/atl/)
Attribute	A piece of information associated with an entity. Used to describe a characteristic of an entity. In the meta-model, an attribute is a placeholder for the actual information that is defined in the models.
CESAR	"CESAR" stands for Cost-efficient methods and processes for safety relevant embedded systems and is a European funded project from ARTEMIS JOINT UNDERTAKING (JU). (See: http://www.cesarproject.eu/)
Domain-specific Language	Domain-specific language (DSL) is a programming language or specification language dedicated to a particular problem domain.
DSL	Domain-specific Language
Closed World Assumption	The closed world assumption is the presumption that what is not currently known to be true, is false. The opposite of the closed world assumption is the open world assumption. (See: http://en.wikipedia.org/wiki/Closed_world_assumption)
CWA	See: Closed World Assumption
Eclipse Modeling Framework	Is a modeling framework and code generation facility for building tools and other applications based on a structured data model. (See: http://www.eclipse.org/emf/)
eCore	Pendant to EMOF in EMF.
EMF	See: Eclipse Modeling Framework
Entity	An object with an identity that is distinguishable from other entities.
Error	Discrepancy between a computed, observed or measured value or condition and the true, specified or theoretically correct value or condition.
Extensible Markup Language	Is a set of rules for encoding documents in machine-readable form defined by W3C. (See: http://www.w3.org/TR/2008/REC-xml-20081126/)
Generic Modeling Environment	Is a configurable toolkit for creating domain-specific modeling and program synthesis environments. (See: http://www.isis.vanderbilt.edu/Projects/gme/)
GME	See: Generic Modeling Environment
Heterogeneous Rich Component	Component describing meta model to facilitate heterogeneous information from different domains (e.g. software development, hardware development) and aspects (e.g. functional, non-functional), first developed in the IP-SPEEDS project

HRC	See: Heterogeneous Rich Component
iFEST	The iFEST project (industrial Framework for Embedded Systems Tools) aims at specifying and developing an tool integration framework for HW/SW co-design of heterogeneous and multi-core embedded systems. (See: http://www.artemis-ifest.eu/)
Instance	An entity that is obtained from another entity by the process of instantiation.
Interface	Abstraction of a service that only defines the operations supported by that service (publicly accessible variables, procedures, or methods), but not their implementation.
IP-SPEEDS	See: Speeds
Link	Representation of a relation between two objects.
Meta-Object Facility	Is an OMG standard for model-driven engineering. (See: http://www.omg.org/mof/)
Meta-Object Facility 2	(See: www.omg.org/spec/MOF/2.0)
Metamodel	A model for describing (structure of) other models on a meta level.
Model	A semantically closed abstraction of a software or hardware system, i.e. a simplification of reality that gives a complete description a system from a particular perspective.
ModelBus	ModelBus is an model driven tool and information integration, process automation and user collaboration framework. (See: www.modelbus.org)
MOF	See: Meta-Object Facility
MOF2	See: Meta-Object Facility 2
MOF Support For Semantic Structures	Is a standardized extension to MOF by the OMG which modifies MOF 2 in order to support dynamically mutable multiple classifications of elements and to declare the circumstances under which such multiple classifications are allowed, required and prohibited. (See: http://www.omg.org/spec/SMOF/)
Object Constraint Language	Is a declarative language for describing rules that apply to MOF compliant metamodels. (See: www.omg.org/spec/OCL/)
Object Management Group	Is a standardization organization in the modelling area (e.g. programs, systems and business processes). (See: http://www.omg.org)
OCL	See: Object Constraint Language
OMG	See: Object Management Group
Open World Assumption	In formal logic, the open world assumption is the assumption that the truth-value of a statement is independent of whether or not it is known by any single observer or agent to be true. Stating that lack of knowledge does not imply falsity. It is the opposite of the closed world assumption. (See: http://en.wikipedia.org/wiki/Open_world_assumption)
Open Services for Lifecycle Collaboration	Open Services for Lifecycle Collaboration (also known as OSLC or Open Services) is a community and set of specifications for Linked Lifecycle Data. The community's goal is to help product and software delivery teams by making it easier to use lifecycle tools in combination. (See: http://open-services.net/html/Home.html)
OSLC	See: Open Services for Lifecycle Collaboration

OWA	See: Open World Assumption
OWL	See: Web Ontology Language
RDF	See: Resource Description Framework
Resource Description Framework	It's a family of W3C specifications for conceptual description or modelling of information that is implemented in web resources. (See: http://www.w3.org/TR/rdf-primer/)
SMOF	See: MOF Support For Semantic Structures
SPARQL	SPARQL (SPARQL Protocol And RDF Query Language) is a query language for RDF. http://www.w3.org/TR/rdf-sparql-query/
Speeds	Speeds (speculative and exploratory design in systems engineering) is an FP7 Project. (See: http://www.speeds.eu.com/)
System	A collection of components organized to accomplish a specific function or set of functions.
Traceability	Traceability is a technique used to provide relationships between objects (e.g. in requirements, design and implementation of a system in order to manage the effect of change).
Type system	A tractable syntactic framework for classifying phrases according to the kinds of values they compute. See: http://en.wikipedia.org/wiki/Type_system
UML	See: Unified Modeling Language
UML Profile	A profile in the Unified Modeling Language provides a generic extension mechanism for building UML models in particular domains. Profiles are based on additional stereotypes and tagged values that are applied to elements, attributes, methods, links, and link ends. A profile is a collection of such extensions and restrictions that together describe some particular modeling problem and facilitate modeling constructs in that domain.
Unified Modeling Language	From Grady Booch, Ivar Jacobson and Jim Rumbaugh (Co. Rational software) developed, and by the Object Management Group (OMG) confirmed abstract description language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for other non-software systems.
View	A representation of a whole system from the perspective of a related set of concerns.
W3C	See: World Wide Web Consortium
Web Ontology Language	Is a family of knowledge representation languages defined by W3C for authoring ontologies. (See: http://www.w3.org/TR/owl-ref/)
World Wide Web Consortium	Is the main international standardization organization for the World Wide Web. (See: http://www.w3.org)
XML	See: Extensible Markup Language
XML Schema	Language for XML schema description which has Recommendation status by the W3C. (See: http://www.w3.org/TR/xmlschema-1/, http://www.w3.org/TR/xmlschema-2/)
XSD	See: XML Schema

6 References

[MOF, 2002] Object Management Group: *Meta Object Facility (MOF) Specification*, Version 1.4 (OMG Document formal/02-04-03), April 2002. 3, 5, 9

[MOF, 2006] Object Management Group: *Meta Object Facility (MOF) Core Specification*, Version 2.0 (OMG Document formal/06-01-01), January 2006. 12

[UML2I, 2003] Object Management Group: OMG ptc/03-09-15:*UML 2.0 Infrastructure Final Adopted Specification*

[QVTRFP, 2002] Object Management Group: MOF 2.0 Query / Views / Transformations RFP, Request for Proposal (OMG Document ad/2002-04-10), April 2002. 12

[JSR040, 2002] Java Metadata Interface(JMI) Specification, June 2002. Java Community Process, JSR 040. 11

[QVT, 2005] Object Management Group: Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Final Adopted Specification (OMG Document ptc/05-11-01), November 2005. 7, 13, 14, 15, 16, 17, 18, 19, 51, 53

[Torgersson, 1996] Olof Torgersson. A Note on Declarative Programming Paradigms and the Future of Definitional Programming. In Das Winteroete'96, 1996. 13

[Czarnecki, 2003] Krzysztof Czarnecki and Simon Helsen. Classification of Model Transformation Approaches. *OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture*, 2003.

[Jouault, 2006] Frédéric Jouault and Ivan Kurtev. On the Architectural Alignment of ATL and QVT. In Proceedings of the 2006 *ACM Symposium on Applied Computing (SAC 06)*, pages 1188–1195, Dijon, France, 2006. ACM Press.

[Kurtev, 2006] Frédéric Jouault and Ivan Kurtev. Transforming Models with ATL. *In Satellite Events at the MoDELS 2005 Conference*, volume Volume 3844/2006. Springer Berlin / Heidelberg, 2006.

[ATLAS-ATL] ATLAS Group. The atlas transformation language (atl). http://www.sciences.univ-nantes.fr/lina/atl.

[ATLAS-AMMA] ATLAS Group. The amma platform. http://www.sciences.univ-nantes.fr/lina/atl/AMMAROOT/

[Wagner, 2005] Michael Wagner. *Evaluation and Implementation of Metamodel-based Transformation Approaches.* Master's thesis, Technische Universitaet Berlin, Faculty IV (Electric Engineering and Computer Science), Department for Formal Models, Logic and Programming (FLP), November 2005. 16, 18, 50

[Koehler, 2003] Jana Koehler Tracy Gardner, Catherine Griffin and Rainer Hauser. A review of OMG MOF 2.0 Query / Views / Transformations Submissions and Recommendations towards the final Standard. In MetaModelling for MDA Workshop, 2003. 13, 41

[Motik, 2007] B. Motik, I. Horrocks, and U. Sattler; *Bridging the gap between OWL and relational databases*; Proceeding WWW '07 Proceedings of the 16th international conference on World Wide Web, pages 807-816. ACM Press, 2007, ISBN: 978-1-59593-654-7

[Baader, 2003] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors; *The Description Logic Handbook: Theory, Implementation and Applications*; Cambridge University Press, 2003 ISBN-13: 978-0521781763

[McGuinness, 2002] McGuinness, D.L. (2002), "Ontologies come of age," in *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*, eds. D. Fensel, J. Hendler, H. Lieberman, & W. Wahlster, MIT Press, Boston, MA, pp. 1–18.

[MOF2] OMG; *MOF Core Specification*; http://www.omg.org/spec/MOF/2.0/PDF (accessed 16.5.2011)