Towards a Comparative Analysis of Meta-Metamodels

Heiko Kern
University of Leipzig
Johannisgasse 26
04103 Leipzig, Germany
kern@informatik.uni-leipzig.de

Axel Hummel
University of Leipzig
Johannisgasse 26
04103 Leipzig, Germany
hummel@informatik.uni-leipzig.de

Stefan Kühne
University of Leipzig
Johannisgasse 26
04103 Leipzig, Germany
kuehne@informatik.uni-leipzig.de

ABSTRACT

A cornerstone in Domain-Specific Modeling is the definition of modeling languages. A widely used method to formalize domain-specific languages is the metamodeling approach. Fortunately, there are a huge number of metamodeling languages. This variety leads to two problems: the selection of a metamodeling language and the interoperability between metamodeling tools. In this paper, we analyze a set of metamodeling languages (ARIS, Ecore, GME, GOPPRR, MS DSL Tools, and MS Visio), define criteria for comparison, and compare the selected meta-metamodels. The comparison forms a foundation for solving the selection and interoperability problem.

Categories and Subject Descriptors

D.3.3 [**PROGRAMMING LANGUAGES**]: Language Constructs and Features

General Terms

Languages

Keywords

Domain-Specific Language, Metamodeling, Comparison

1. INTRODUCTION

Domain-Specific Modeling (DSM) is a software development approach that uses models as primary artifacts in the development process [8]. Models are an abstraction of a concrete implementation and are expressed by a domain-specific modeling language. A cornerstone in the development of a DSM architecture is the definition of languages. In the area of DSM, the metamodeling approach is a widely used method to formalize languages with the help of metamodels. Generally speaking, a metamodel defines the concepts of a modeling language and their relationships as well as constraints and modeling rules. To define a metamodel, a metamodeling language is required that in turn is described by a meta-metamodel [16].

Fortunately, there are a huge number of metamodeling languages. However, this variety also implies problems. The first problem relates to the selection of a metamodeling language. For a language designer, the choice of a suitable metamodeling approach is a challenging task because there is often a lack of knowledge about the selection criteria and the offered metamodeling features. The second problem concerns the model interoperability. The development of a complex modeling infrastructure or the replacement of a metamodeling tool requires the transformation of metamodels and models between these metamodeling tools. An example for this problem is the bridging transformation (e.g. [3, 4, 6, 9, 10, 11]). The implementation of a bridge requires knowledge about metamodeling concepts and their possible mappings. We address these issues by comparing different meta-metamodels. Particularly, our objective is a comparative analysis of meta-metamodels in order to extract their basic language features with possible variation.

The paper is structured as follows. In the subsequent section, we give an overview of the meta-metamodels included in the comparison. In Section 3 we describe metamodeling concepts and compare the selected meta-metamodels. In Section 4 we evaluate the comparison and in Section 5 we relate our work to other comparison approaches. Finally, we conclude in Section 6 with a discussion and describe future work.

2. META-METAMODELS UNDER STUDY

We focus our comparison on a selected set of metamodeling languages that fulfill the following requirements. The first requirement refers to the kind of language definition. We can distinguish two approaches: the lightweight and heavyweight approach [7]. The lightweight variant adapts a generic metamodel with domain-specific concepts. An example for this approach is the stereotype mechanism from the Unified Modeling Language [13]. The heavyweight approach creates a language through the definition of metamodels which are created from scratch. In this article, we investigate the heavyweight approach with a three-level hierarchy that consists of models, metamodels and one metametamodel. A further requirement concerns the concrete syntax. If we distinguish between textual and graphical modeling languages, we select metamodeling languages that enable the definition of graphical languages with textual annotations. The last requirement is the availability as tool. Based on these requirements, we choose the following tools or meta-metamodels, respectively.

ARIS The Architecture of Integrated Information Systems [15] is an approach to enterprise modeling. The associated tool supports by default different modeling notations. Users can adapt already existing languages and the vendor can create completely new languages. The ARIS meta-metamodel was analyzed in [10] and is shown in Figure 1(a).

Ecore Ecore [2] is the meta-metamodel in the Eclipse Modeling Framework. The framework supports the development of (Eclipse) applications. Figure 1(b) shows the main concepts. Ecore has a strong relation to the Essential Meta Object Facility [14]. There are many model processing tools based on Ecore.

GME The Generic Modeling Environment [12] is primarily a tool for domain-specific modeling in the area of electrical engineering. The definition of languages based on the meta-metamodel that is shown in Figure 1(c).

GOPPRR Graph, Object, Port, Property, Role, Relationship [8] is the metamodeling language in MetaEdit+. The tool supports typical DSM tasks such as (meta) modeling and code generation. The meta-metamodel is shown in Figure 1(d)).

MS DSL Tools The Microsoft Domain-Specific Language Tools enables the definition of languages and generators into Visual Studio. The metamodeling is described implicitly in [5]. We give an explicit description in form of an meta-metamodel that is shown in Figure 1(e).

MS Visio Microsoft Visio [1] is an universal modeling and data visualization tool. Language definitions are realized by stencils. We analyzed the stencil definition and extract the meta-metamodel in [11] (see Figure 1(f)).

In our comparison we use the following tool versions: ARIS Business Architect 7.1, EMF Ecore 2.5, GME 10.8, Meta Edit+ 4.5, MS DSL Tools in Visual Studio 2008, and MS Visio 2007. Furthermore, all meta-metamodels in Figure 1 are shown as UML Class Diagrams.

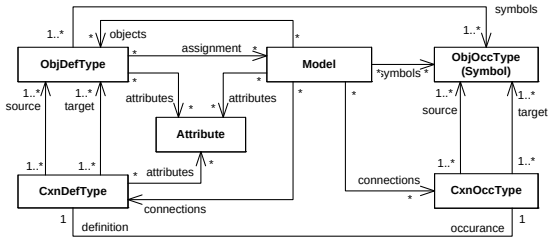
3. METAMODELING CONCEPTS

In this section, we present our comparison framework. We identify first class metamodeling concepts available in each meta-metamodel and abstract these concepts to comparison criteria such as object, relationship, role, port, model, and attribute. Based on these criteria, we analyze further concept-specific properties and possible values. Additionally, we investigate concepts for structuring, reusing, and identification of metamodel elements. Table 1 shows the results of the comparison.

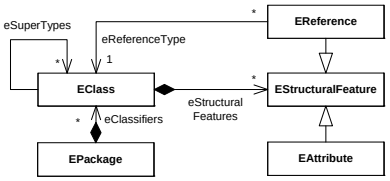
Object Type An object type defines a set or a class of objects with equal features such as relationship types or attributes. In general, an object type represents a concept or entity type of a modeling domain.

Relationship Type A relationship type connects object types and is defined as a subset of the (n-ary) Cartesian product over the participating object types. A relationship type can have the following properties.

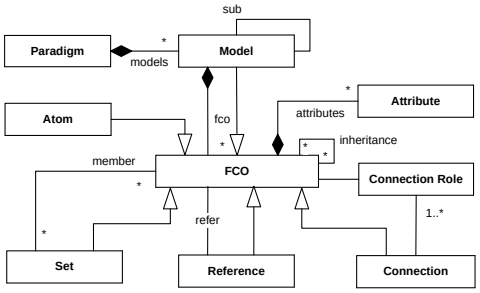
- *Arity:* The arity specifies the number of object types (or role types) that can be involved in a relationship type. Possible values are binary or n-ary relationship types.



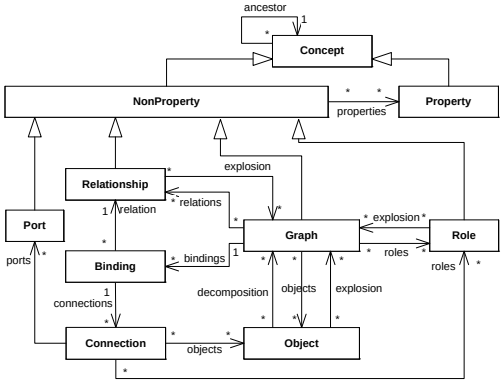
(a) ARIS



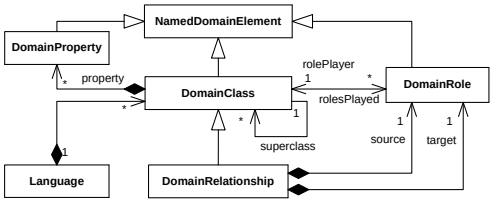
(b) Ecore



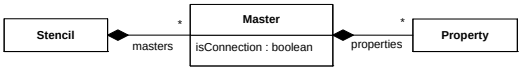
(c) GME



(d) GOPPRR



(e) MS DSL Tools



(f) MS Visio

Figure 1: Overview of Meta-Metamodels

- *Multiplicity:* The multiplicity defines the number of objects that can be related to another object in context of a relationship type. The multiplicity can be defined with concrete values/intervals (e.g. n, 0..1, 0..n, 1..n). If the multiplicity is not supported, than the value is often 0..n by default ($n \in N$).
- *Inverse:* A binary relationship type can be defined as inverse of another relationship type. The inverse relationship contains (automatically) the tuples of the original relationship in opposite order.
- *Composition:* A composition influences the life cycle dependency between a container object and the contained objects which are hold in a relationship. Generally, if the container object is destroyed, every related/contained object is destroyed as well.
- *Dependency:* The dependency influences the life cycle of a relationship type. Generally, a relationship type may be dependent on an object type or other elements.
- *Object-Set:* Normally, a relationship type assigns one object type for each end. This feature enables the connection of an object type set for each end.
- *Role:* A relationship type can support a role concept.

Role Type A role type defines how object types participate in a relationship type. If the meta-metamodel supports a role concept, then a relationship references an object indirectly over a role. Thus, some features of a relationship type are transferred to role type.

- *Multiplicity:* This feature is taken from the relationship type. The multiplicity is defined in the role type rather than in the relationship type.
- *Dependency:* This feature is analog to relationship type. A role type may be dependent on a relationship type or other elements.
- *Object-Set:* This feature is also analog to relationship type. Each role type relates to one object type. This feature enables the connection of an object type set for each end.

Port Type A port type defines a set or a class of ports. A port allows additional semantics or constraints on how objects can be connected by a relationship.

Model Type A model type is a set or a class of models with equal properties. A model type groups available metamodel elements and is instantiable.

Links to Model Types Some meta-metamodels support links between model types and other metamodel elements. This concept enables, for instance, the refinement of model elements or the integration of different model types. The relationship to submodels is binary.

- *Participating elements:* Defines the source and target of the submodel relationship. We identified that the target is a model type and the source are metamodel elements such as object, relation, or role type.
- *Multiplicity:* Defines the number of element instances connected with a submodel relationship. Possible values are one-to-one and one-to-many.

Attribute An attribute is a property of a metamodel element. At model level an attribute can hold concrete values.

- *Data type:* A data type defines the co-domain of an attribute. We can distinguish two kinds of data types: simple data types and other metamodel elements. (1) Simple data types are, for instance, integer, string, or date. (2) If other metamodel elements are possible values, then this kind of attribute is equal to the relationship concept.
- *Dependency:* Attributes are dependent on other elements in the metamodel. Attributes can directly depend on other types. If an attribute is independent from other types, then they can be reused in other metamodeling elements.
- *Multiplicity:* This feature allows to define multi-value attributes, if the multiplicity is greater than one.
- *Ordered:* If the multiplicity is greater than one, the values can be sorted.
- *Unique:* If the multiplicity is greater than one and an attribute is marked as unique, no value may occur multiple times.
- *Default value:* Allows the definition of a default value, if the metamodel element is instanciated.
- *Attributable:* Specifies the attributable metamodel elements such as object, relationship, port, role, or model type.

Inheritance Inheritance is a special relationship between metamodeling elements. In context of metamodeling, inheritance allows to reuse attributes, constraints, or behavior of an (super) element.

- *Participating elements:* Defines the metamodel elements which can participate in an inheritance. These elements can be object, relation, role, or model types.
- *Single vs. multiple inheritance:* A meta-metamodel can support single inheritance, that is, a (sub)element inherits from one (super)element. If multiple inheritance is available, then a (sub)element can inherit from several (super)elements.

Grouping This concept allows to structure metamodel elements in defined parts or modules.

Identification Generally, all metamodel elements have an identifier. This can be a name or number. The scope of identifiers can differ. Some meta-metamodels support multiple identifiers.

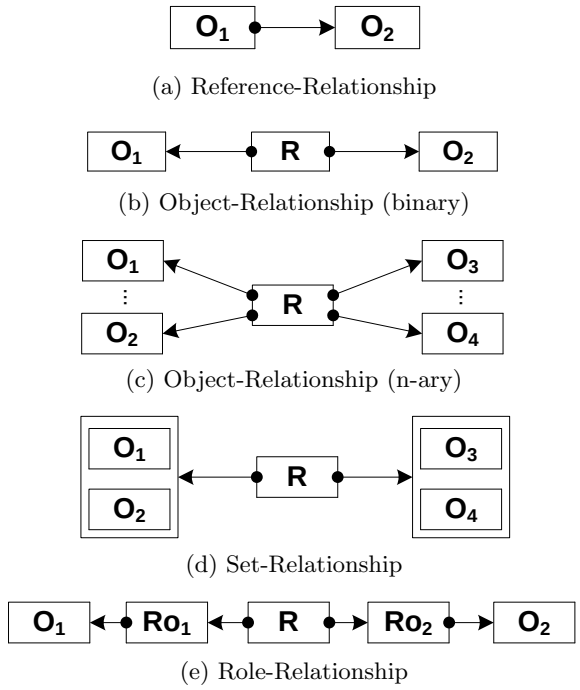
Constraint Language Constraints allow the additional definition of conditions for metamodel elements which have to be fulfilled during or after modeling.

Table 1: Comparison of Meta-Metamodels ((y)es = supported, (n)o = unsupported, – = impossible)

	ARIS	Ecore	GOPPRR	GME	MS DSL Tools	MS Visio
First Class Concepts						
Object	ObjDef, ObjOcc	EClass	Object	Atom, Model, Set	Domain Class	Master
Relationship	CxnDef, CxnOcc	EReference	Relation, (Collection)	Connection, (Set Membership, Reference)	Domain Relationship	Master
Role	n	n	Role	Connection Role	Domain Role	n
Port	n	n	Port	Reference Port	n	n
Model	Model	n	Graph	Model	n	n
Attribute	Attribute	EAttribute	Property	Attribute	Domain Property	Property
Relationship						
Arity	binary	binary	n-ary	binary	binary	binary
Multiplicity	default (0..n)	y	–	–	–	default (0..n)
Inverse	n	y	n	n	n	n
Composition	n	y	n	y	y	n
Dependency	Method	EClass	Project	Paradigm	Domain Class	Stencil
Object-Set	y	n	–	–	–	n
Role						
Multiplicity	–	–	y	y	y	–
Dependency	–	–	Project	Connection	Domain Relationship	–
Object-Set	–	–	y	n	n	–
Attribut						
Dependency	Method	EClass	Project	Paradigm	Domain Class, Domain Relationship	Master
Multiplicity	single-value	multi-value	multi-value	single-value	single-value	multi-value
Unique	–	y	y	–	–	n
Ordered	–	y	n	–	–	y
Default value	n	y	y	y	y	y
Attributable						
Object	y	y	y	y	y	y
Relationship	y	n	y	y	y	y
Role	–	–	y	n	n	–
Port	–	–	y	n	–	–
Model	y	–	y	y	–	–
Data type						
Simple	y	y	y	y	y	y
Metamodel element	n	n	y	n	n	n
Inheritance						
Single/Multiple	n	multiple	single	multiple	single	n
Inheritable						
Object	–	y	y	y	y	n
Relationship	–	n	y	y	n	n
Role	–	–	y	n	n	–
Port	–	–	y	n	–	–
Model	–	–	y	y	–	–
Links to Models	y	–	y	y	–	–
Grouping	Method, Model	EPackage	Project, Graph	Folder, Paradigm	Language, Namespace	Stencil
Constraint Language	n	OCL	proprietary	OCL dialect	programming language	n

Table 2: Supported Relationship Patterns

	ARIS	Ecore	GOPPRR	GME	MS DSL Tools	MS Visio
Reference-Relation (a)		x	x	x		
Binary Object-Relation (b)	x		x	x	x	(x)
Set-Relation (d)	x		x			
Role-Relation (e)			x	x	x	
N-ary Object-Relation (c)			x			



(a) Reference-Relationship

(b) Object-Relationship (binary)

(c) Object-Relationship (n-ary)

(d) Set-Relationship

(e) Role-Relationship

Figure 2: Relationship Patterns (O = Object Type, R = Relationship Type, Ro = Role Type, → = Reference)

4. EVALUATION

Based on the comparison, we evaluate some results from Table 1. The first aspect refers to metamodeling concepts available as first class elements. Object type, relationship type and attribute are the fundamental metamodeling concepts because each meta-metamodel supports these concepts. This observation is consistent with the statement from Kelly and Tolvanen [8]. Additionally, the half of all meta-metamodels supports the concepts model type and role type. Model type allows the implementation of a view concept and role type enables the realization of complex relationships. In dependency of the supported first class modeling concepts, we can deduce the following order of meta-metamodels[1]: GOPPRR ≡ GME > MS DSL Tools > ARIS > Ecore ≡ Visio.

The second aspect focuses on the definition of relationships. We identified five kinds of relationships which are shown in Figure 2. (a) The *reference-relationship* can be interpreted as a simple reference or pointer between object types. This relationship is binary, depends on an object type, and is not attributable. (b) In contrast to the reference-relationship, the *binary object-relationship* can be considered as a "stand-alone" modeling concept because the life cycle is often independent from an object type. Furthermore, this relationship is binary and attributable. (c) The *n-ary object-relationship* is analogous to the binary object-relation with the difference that this relation allows to reference more than two object types. (d) Based on the object-relationship, the *set-relationship* allows to connect a set of object types for each relation end. The interpretation of this relation is the Cartesian product between both object type sets. (e) The *role-relationship* uses an explicit role concept that is often attributable. An assignment of the different relationships to the meta-metamodels is shown in Table 2. If we count the number of supported relationship patterns and assume the order of relationships suggested in the left column in Table 2, we can derive the following ranking[1]: GOPPRR > GME > DSL Tools > ARIS > Ecore > Visio.

The last aspect refers to the concepts for structuring, reuse and modularization in metamodeling. Four out of six meta-metamodels support inheritance. Meta-metamodels with inheritance are Ecore, GOPPRR, GME, and DSL Tools. Inheritance refers primarily on the reuse of attributes and modeling constraints. Generally, the four meta-metamodels enable inheritance between object types. GOPPRR and GME realize inheritance additionally between elements such as relationship or model type. ARIS and MS Visio support no inheritance. Furthermore, all meta-metamodels offer a concept for grouping metamodeling elements. This can be the model concept or other container elements such as Project, Folder, Stencil, Language, or EPackage.

[1] $x \equiv y$: the number of metamodeling concepts in x and y is equal, $x > y$: x supports more metamodeling concepts than y

5. RELATED WORK

The comparison of meta-metamodels has been the subject of previous investigations. A relevant contribution to our work comes from Kelly and Tolvanen [8]. Based on their experience in the area of metamodeling, the authors propose basic metamodeling concepts. These concepts are, for instance, explicit graph type, explicit role concept, n-ary relationship, or links to subgraphs. However, their comparison of tools (MetaEdit+, MS DSL Tools, Eclipse Modeling Project, and GME) are more general nature. Furthermore, in a prior work Tolvanen [17] compares seven metamodeling languages (ASDM, CoCoA, ER, GOPRR, MEL/MDM, NIAM, OPRR) against criteria such as unique property, data type of properties, multiplicity, explosion or polymorphism. The aim of this comparison was to find a metamodeling language that fits to the defined requirements in the context of method engineering. The difference to our comparison is the underlying method. Kelly and Tolvanen [8, 17] use a top-down-approach because they suggest concepts, that are required for suitable metamodeling. In our work, we use a bottom-up approach. We investigate a set of metamodeling languages and abstract from specific features occurring in different meta-metamodels. The decision in the

abstraction process is influenced by the top-down approach from Kelly and Tolvanen [8].

Further contribution comes from the transformations between different meta-metamodels. Basis of a transformation is the knowledge about the semantic of the different metamodeling concepts because the transformation should preserve the semantics. There are the following bridging transformations: GME and Ecore [3], MS DSL Tools and Ecore [4], MOF1.4 and Ecore [6], ARIS and Ecore [10], MetaEdit+ and Ecore [9], and Visio and Ecore [11]. In relation to our work, the transformation between the meta-metamodels has a limited focus because each paper compares only two meta-metamodels. Thus, the comparison is very specific. In our work, we try to abstract from very specific features in meta-metamodels in order to find an unified comparison framework.

6. CONCLUSION

In this paper, we compared six meta-metamodels that are using the heavyweight metamodeling approach and are available as tools. We analyzed each meta-metamodel and extracted typical metamodeling features. Based on this extraction, we defined a comparison framework. The result of this comparison is shown in Table 1. In general, all investigated meta-metamodels have the basic elements: object, relationship, and attribute. In particular, we have found differences in relationships and supported features for structuring and reuse of metamodel elements.

Regarding the language selection problem in the introduction, the comparison helps to get an overview of the concepts supported in the respective metamodeling languages. A suitable indicator for the language selection is the practical expressiveness. We assume the practical expressiveness of meta-metamodels is direct proportional to the number of supported metamodeling concepts. Based on this assumption and the evaluation in Section 4, we conclude that GOPPRR and GME are the most powerfully expressive meta-metamodels and Visio has the least expressive power of all metamodeling languages. However, other criteria (e.g. usage, standardization, or tool features) play a crucial role for the selection of a certain metamodeling approach. Relating to the interoperability problem, the comparison enables the development of abstract transformation rules between metamodeling concepts. In order to specify such rules, we need a theoretical/mathematical foundation of the extracted metamodeling concepts. The theoretical foundation and the development of possible transformations between these abstract concepts are future work.

7. REFERENCES

[1] B. Biafore. *Visio 2007 Bible*. John Wiley & Sons Inc., 2007.

[2] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, and T. J. Grose. *Eclipse Modeling Framework*. The Eclipse Series. Addison Wesley, 2004.

[3] J. Bézivin, C. Brunette, R. Chevrel, F. Jouault, and I. Kurtev. Bridging the Generic Modeling Environment (GME) and the Eclipse Modeling Framework (EMF). In *Proceedings of the Best Practices for Model Driven Software Development at OOPSLA'05*, San Diego, California, USA, 2005.

[4] J. Bézivin, G. Hillairet, F. Jouault, I. Kurtev, and W. Piers. Bridging the MS/DSL Tools and the Eclipse Modeling Framework. In *Proceedings of the International Workshop on Software Factories at OOPSLA 2005*, San Diego, California, USA, 2005.

[5] S. Cook, G. Jones, S. Kent, and A. C. Wills. *Domain Specific Development with Visual Studio DSL Tools (Microsoft .Net Development)*. Addison-Wesley Longman, 2007.

[6] K. Duddy, A. Gerber, and K. Raymond. Eclipse Modelling Framework (EMF) import/export from MOF / JMI. Technical report, CRC for Enterprise Distributed Systems Technology (DSTC), 2003.

[7] D. S. Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing*. John Wiley & Sons, 2003.

[8] S. Kelly and J.-P. Tolvanen. *Domain-Specific Modeling: Enabling Full Code Generation*. John Wiley & Son, Inc., 2008.

[9] H. Kern. The Interchange of (Meta)Models between MetaEdit+ and Eclipse EMF Using M3-Level-Based Bridges. In J.-P. Tolvanen, J. Gray, M. Rossi, and J. Sprinkle, editors, *8th OOPSLA Workshop on Domain-Specific Modeling at OOPSLA 2008*, 2008.

[10] H. Kern and S. Kühne. Model Interchange between ARIS and Eclipse EMF. In J.-P. Tolvanen, J. Gray, M. Rossi, and J. Sprinkle, editors, *7th OOPSLA Workshop on Domain-Specific Modeling at OOPSLA 2007*, 2007.

[11] H. Kern and S. Kühne. Integration of Microsoft Visio and Eclipse Modeling Framework Using M3-Level-Based Bridges. In *2nd ECMDA Workshop on Model-Driven Tool and Process Integration, 24 June 2009, at Fifth European Conference on Model-Driven Architecture Foundations and Applications 2009*, Enschede, Netherlands, 2009.

[12] A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle, and P. Volgyesi. The Generic Modeling Environment. In *IEEE International Workshop on Intelligent Signal Processing*, 2001.

[13] Object Management Group. *Unified Modeling Language: Infrastructure, version 2.0*, 2006.

[14] Object Management Group. *Meta Object Facility (MOF) Core Specification*, 2010.

[15] A.-W. Scheer. *ARIS: Business Process Modeling*. Springer, 2000.

[16] T. Stahl and M. Völter. *Model-Driven Software Development*. John Wiley & Sons Inc., 2006.

[17] J.-P. Tolvanen. *Incremental Method Engineering with Modeling Tools: Theoretical Principles and Empirical Evidence*. PhD thesis, University of Jyväskylä, 1998.