The Interchange of (Meta)Models between MetaEdit+ and Eclipse EMF Using M3-Level-Based Bridges

Heiko Kern

Business Information Systems, University of Leipzig
Johannisgasse 26, 04103 Leipzig, Germany
kern@informatik.uni-leipzig.de

Abstract

Nowadays there are powerful tools for Domain-Specific Modeling. An ongoing problem is the insufficient tool interoperability which complicates the development of complete tool chains or the re-use of existing metamodels, models, and model operations.

In this paper we present the approach of M3-Level-Based Bridges and apply this approach to enable the interoperability between two selected tools. The first tool is MetaEdit+ with strengths in (meta)modeling and the second tool is the Eclipse Modeling Framework with advantages in model processing by transformation and generation tools.

General Terms Model-Driven Engineering, Metametamodel, Domain-Specific Language, Interoperability

Keywords Tool Interoperability, MetaEdit+, Eclipse Modeling Framework, Model Transformation, M3-Level-Based Bridge

1. Introduction

Domain-Specific Modeling (DSM) is a software development approach which basically uses the following two concepts. First, DSM applies special-purpose languages covering the domain concepts of the problem space. Second, DSM uses model operations such as generators or transformations working on domain-specific models to automate the development of executable software (Kelly and Tolvanen 2008). Similar to other development approaches, the tool support is a crucial factor for the success of DSM. Fortunately, powerful tools are available that support typical DSM tasks such as language definition, modeling, and model processing. Some of these tools are, the Microsoft DSL Tools (Cook et al. 2007), MetaEdit+ (MetaCase 2008), Generic Modeling Environment (Ledeczi et al. 2001), Eclipse Graphical Modeling Framework (GMF 2008), openArchitectureWare (oAW 2008) or the AMMA Platform (AMM 2008).

But an ongoing problem is the insufficient tool interoperability. This lack of interoperability complicates the development of complete tool chains or the re-use of existing metamodels, models, and model operations (Bézivin et al. 2005a; Karsai et al. 2003). In this article, we present the approach of M3-Level-Based Bridges to achieve tool interoperability between different DSM tools. We apply this approach to realize the interchange of metamodels and models between two selected tools: MetaEdit+ (abbreviated MetaEdit) and the Eclipse Modeling Framework (abbreviated Eclipse EMF).

MetaEdit from MetaCase is a widely-used commercial DSM tool which supports the developer during the definition of Domain-Specific Languages (DSLs) and allows modeling with these DSLs. Further, the tool includes a code generator. In addition to the DSM functionality, MetaEdit provides an extensive model repository with multi-user functionality.

The second tool that supports typical DSM tasks is given by the Eclipse Modeling Framework and by tools based on this framework. Eclipse EMF allows the definition of domain-specific metamodels and provides basic functionality such as an API to access (meta)models, a serialization function to XMI/XML, and a generator language. Based on Eclipse EMF, the selection of tools is diverse, ranging from model transformation tools (Jouault and Kurtev 2005; Lawley and Steel 2005)and tools for DSL modeling (GMF 2008; Grundy et al. 2008)to other model processing tools (Fabro et al. 2005; EMF 2008).

The approach of M3-Level-Based Bridges will be applied on MetaEdit and Eclipse EMF because both tool spaces can benefit from the created interoperability. While MetaEdit has advantages in (meta)modeling, Eclipse EMF provides a wide range of model processing tools.

The paper is structured as follows: in the subsequent section, we will give a conceptual overview of M3-Level-Based Bridges. As the bridge is based on the mapping between metametamodels, we will explore the metamodeling language of MetaEdit and the metametamodel of Eclipse EMF in section 3.1 and 3.2, respectively. In section 4 we will present the development of the bridge. In section 5 we will describe the implementation of the bridge and will show an example in section 6. Lastly, we will summarize the article and will conclude with future challenges.

2. M3-Level-Based Bridges

2.1 Conceptual Overview

The idea of M3-Level-Based Bridges is to achieve interoperability between different tools by transforming models and metamodels. This approach is well-established and has been successfully applied in building bridges between MetaGME and Eclipse EMF (Bézivin et al. 2005b), Microsoft DSL Tools and Eclipse EMF (Bézivin et al. 2005c), ARIS Toolset and Eclipse EMF (Kern and Kühne 2007), or Meta Object Facility and Eclipse EMF (Duddy et al. 2003). Although the implementation of such bridges can differ in technical terms, the conceptual approach is the same.

A prerequisite to construct bridges is the existence of a metamodel hierarchy (Kühne 2006; Atkinson and Kühne 2003; Gitzel and Hildenbrand 2005) consisting of three levels. At the lowest level (M1-level) are models which describe a software system. The structure of these models and the

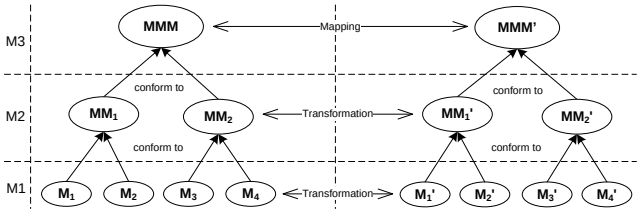

Figure 1. Conceptual Approach of M3-Level-Based Bridges.

available concepts that can be instantiated in models are defined by a metamodel at M2-level. Finally the structure and the available concepts of the metamodels are defined by a metametamodel at M3-level. Such a M3-level hierarchy often occurs in DSM tools because it easily enables the development of DSLs.

Based on the existence of such hierarchies, the basic step to build M3-Level-Based Bridges is the mapping between the metametamodels (see Fig. 1). The mapping consists of different mapping rules specifying the relation between semantically equivalent concepts. Semantically equivalent means, for instance, that concepts at M3-level expressing relationships at M2-level are mapped onto each other. Based on the mapping specification the transformation of metamodels and models can be derived. To create the transformations, it is necessary to know how the instance relationship is realized between each level. The M2-level transformation maps metamodels between hierarchies. These metamodels are isomorphic. Analogous to the M2-Level transformation, the M1-level transformation enables the mapping of models. These models are also isomorphic.

2.2 Typical Metamodeling Concepts

The M3-level mapping is the basic concept of M3-Level-Based Bridges. Many metametamodels have similar concepts for metamodeling that can be mapped on each other. In this section we want to describe these typical metamodeling concepts (i. e. a kind of equivalence classes for metamodeling concepts) to provide an assistance for the latter M3-level mapping between MetaEdit and Eclipse EMF (in Sec. 4.2). Moreover, these general concepts can be helpful to build other M3-level mappings.

We have analyzed the following metametamodels: Ecore from Eclipse EMF (Budinsky et al. 2004), GOPRR from MetaEdit (Kelly and Tolvanen 2008; Kelly 1997; Tolvanen 1998), A3 from ARIS Toolset (Kern and Kühne 2007), MOF (version 1.4) (Obj 2002), and the metametamodel from Microsoft DSL Tools (Bézivin et al. 2005c; Cook et al. 2007). As a result, we suggest the following classes of metamodeling concepts:

Object type An object type is a concept to define a set or a class of objects with equal features. Other names for object type can be class, metaclass, entity, or object.

Relation type A relation type defines a set of relations between objects. Other notations are relationship, reference, connection, or association.

Attribute type Attribute types define features for metamodel elements such as object types or relation types. Another name can be property type.

Data type A data type specifies the range of values in attributes. Typical data types are, for instance, `integer`, `string`, or `date`. But it can also be object types or relationship types.

Model type A model type is a concept to define a set of models consisting of defined metamodel elements. Other notations are graph type, domain model or metamodel.

Inheritance This concept can be used to relate metamodel elements in a inheritance relationship. Usually this concept can be interpreted like the inheritance in object orientation programming languages.

Partitioning This concept allows the (logical) structuring of metamodels in defined parts. Other names can be namespace or package.

Depositation A depositation often specifies a relation between metamodel elements (such as object type or relation type) and model types. Other designations are explosion, decomposition or assignment.

Constraint Constraints enable the specification of conditions which have to be fulfilled during or after modeling.

The results of the study are intentionally abstract because a detailed metametamodel comparison is not the focus of this paper.

3. (Meta)Modeling with MetaEdit and Eclipse EMF

3.1 Domain-Specfic Modeling in MetaEdit

MetaEdit is an established DSM environment which provides a powerful language for metamodeling. The concepts of this (meta)language are defined in the GOPRR model. GOPRR is the abbreviation for Graph, Object, Property, Role and Relationship and is shown in Figure 2.

A graph type (instance of `Graph`) specifies a model type that contains the modeling concepts: object types (instances of `Object`), relationship types (instances of `Relationship`), and role types (instances of `Role`). An object type describes a class of model elements which can exist on their own. Object types can be connected by relationship types, whereby the role type specifies how an object type participates in a relationship type. Relations can be defined by the concept of `Binding`. A binding connects a relationship type, two or more role types, and one or more object types for each role type in a graph type. Each previously described language concept, except for the `Binding` concept, can have property types (instances of `Property`) to characterize language concepts. The values of property types can be different data types such as a string, text, number, boolean, collection, or a link to other modeling language concepts. Other concepts supported in GOPRR are `Inheritance` that makes the creation of subtypes possible, `Decomposition` which enables object types to have subgraph types, and `Explosion` which allows object types, relationship types, or role types to be linked to other graph types.

The notation or concrete syntax of a modeling language can be defined with the help of a symbol editor. The editor supports the creation of symbols for object types, relationship types, and role types. It is possible to place a textual representation of property types in symbols.

After describing the language definition, we need to know the structure of the model repository in order to build the bridge. “GOPRR has been designed to be applicable in the

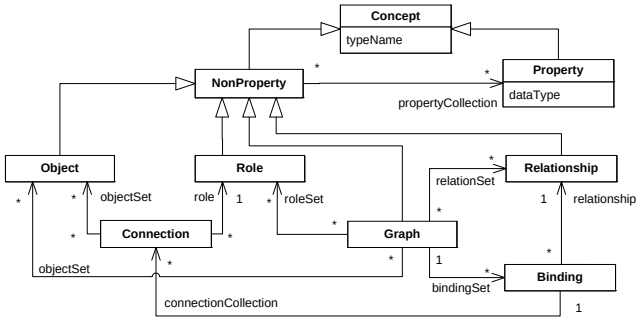

Figure 2. Snippet of GOPRR (Kelly 1997).

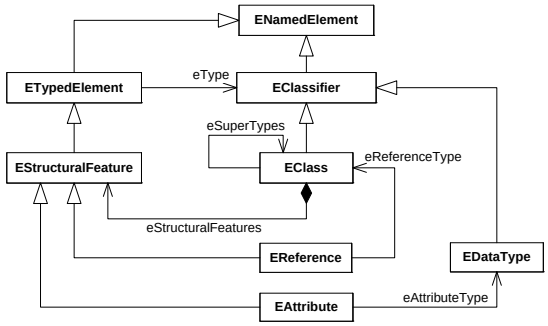

Figure 3. Snippet of Ecore.

same way on both the type and instance levels [...]" (Kelly 1997). That is, the model repository (instance level) is almost equal with the GOPRR model shown in Figure 2. A graph (instance of a graph type) owns a set of relationships (instances of a relationship type), objects (instances of an object type), roles (instances of a role type), and bindings (instances of a binding). A binding stores the combination of a relationship connected with objects and roles. Furthermore, all property values are stored in a value field of the property (instance of property type). All model and metamodel elements are stored in projects. Thus, the GOPRR model contains a project class referencing a graph set.

3.2 Metamodels and Models in Eclipse EMF

Contrary to MetaEdit, developed especially for DSM, Eclipse EMF is designed to support the development of (Eclipse) applications. One reason for this statement is the missing of special metamodeling concepts such as `Model type` or `Depositation` in Ecore. Models (data models) describing Eclipse EMF applications can be regarded as metamodels and instance data of these metamodels as models.

A simplified subset of Ecore is shown in Figure 3 and a detailed description of Ecore is given in Budinsky et al. (2004). The main elements of Ecore are `EClass`, `EReference` and `EAttribute`. An EClass (instance of `EClass`) defines an EMF metamodel element that represents a set of similar model entities. EClasses can have EReferences (instances of `EReference`) which express unidirectional relationships between two EClasses. An EClass can additionally have EAttributes (instances of `EAttribute`) to express properties of the EClass. The range of the attribute values are specified by a data type such as int, string or date. A further metamodeling concept is the inheritance between EClasses.

Analogous to the structure of the MetaEdit model repository, we need to know the structure of the model repository in order to build the M1-level transformation. Every model element in Eclipse EMF is an EObject specified by a Java interface. The EObject interface provides different methods which enable the navigation in models and allow the query of metatype information. For instance, the EObject.eGet(EStructuralFeature) method returns either all EObjects referenced by a certain EReference or values of the EAttributes. Further, the EObject.eClass() method returns the EClass of a model element.

4. MetaEdit to Eclipse EMF Bridge

4.1 Overview of the Bridge

MetaEdit and Eclipse EMF can be structured into three levels: M3 (GOPRR and Ecore), M2 (GOPRR metamodels and EMF metamodels), and M1 (GOPRR models and EMF models). Based on this level structure, we apply the approach of M3-Level-Based Bridge (see Fig. 4).

The M3-level mapping specifies a unidirectional mapping from GOPRR to Ecore. Using this mapping, we can derive the transformation rules at M2-level which export MetaEdit metamodels to Eclipse EMF. Several transformation rules map different GOPRR concepts onto one Ecore concept. To distinguish the different GOPRR concepts in Ecore, we introduce an abstract EMF metamodel (see Fig. 5) that approximates the structure of GOPRR. All exported metamodel elements are inherited from a corresponding abstract metamodel element.

For our purpose, the export of MetaEdit metamodels is sufficient and we do not consider importing EMF metamodels to MetaEdit. But this would also be possible on the condition that all metamodel elements are inherited from an abstract metamodel element.

Based on the M3-level mapping and M2-level transformation, we can derive the M1-level transformation which enables the export and re-import of MetaEdit models.

4.2 M3-Level Mapping

We propose the following mappings:

`Object` $\mapsto$ `EClass`**:** In MetaEdit an object type can define a set of model entities. The corresponding concept in Ecore is the `EClass` concept which can also define a set of model entities. Therefore, we map `Object` onto `EClass`. The `typename` of `Object` maps onto `name` of `EClass`.

`Relationship` $\mapsto$ `EClass`**:** A relationship type between object types is expressed by `Relationship` in GOPRR. In Ecore the `EReference` concept can be used to describe relationship types between EClasses. The problem is that relationship types in MetaEdit can have property types. But EReferences cannot have their own EAttributes. Hence, we cannot map `Relationship` onto `EReference`, but we can map `Relationship` onto `EClass` whereby `typename` of `Relationship` maps onto `name` of `EClass`.

`Role` $\mapsto$ `EClass`**:** A Role type defines how an object type takes part in a relationship type. No direct equivalent concept exists in Ecore. Hence, we map `Role` onto the `EClass` concept. The `typename` of `Role` maps onto `name` of `EClass`.

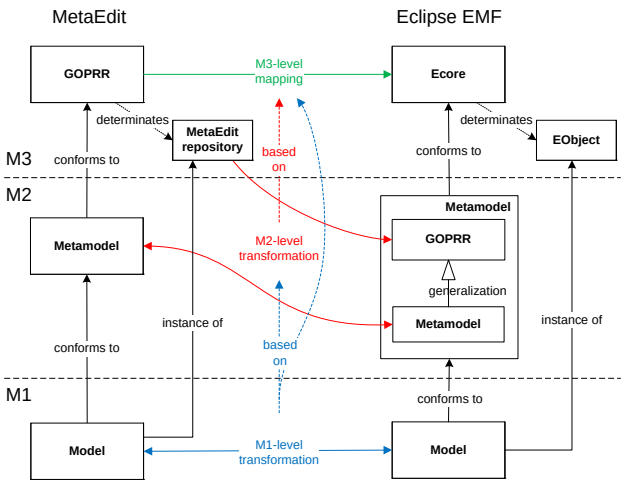

Figure 4. Overview of the MetaEdit to EMF Bridge.

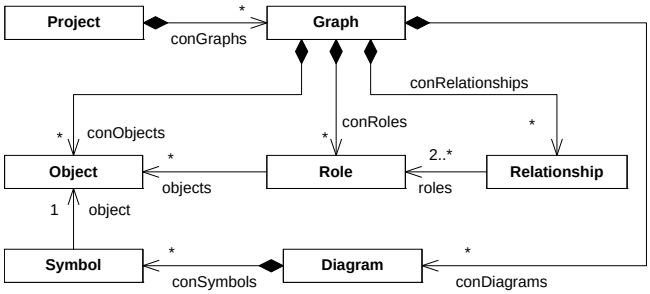

Figure 5. Simplified GOPRR as EMF Metamodel.

`Graph` ↦ `EClass`**:** An equivalent concept for `Graph` does not exist in Ecore. Hence, we map a `Graph` onto `EClass` whereby the `typename` maps onto `name`.

`Property` ↦ {`EAttribute`, `EReference`}**:** We map the `Property` concept onto `EAttribute` or `EReference`. The decision whether it is an `EReference` or an `EAttribute` depends on the data type of a property type. If the data type is a simple type such as string, int or text, then we map `Property` onto `EAttribute`. If the data type is a link to other modeling concepts, then we map `Property` onto `EReference`. The `typename` maps onto `name`.

`Inheritance` ↦ `eSuperType`**:** We map `Inheritance` onto `eSuperType` whereby the related metamodel elements are equal.

4.3 M2-Level Transformation

The M2-level transformation consists of transformation statements which are derived from the M3-level mapping specification. For instance, the mapping rule `Object` ↦ `EClass` transforms all object types in EClasses. These EClasses are inherited from the abstract EClass 'Object' (see Fig. 5) because we need to distinguish those from other EClasses. A further mapping rule `Relationship` ↦ `EClass` transforms all relationship types in EClasses which are all inherited from the abstract EClass 'Relationship'.

4.4 M1-Level Transformation

The M1-level transformation consists of transformation statements which are also derived from the M3-level mapping specification. For instance, the mapping rule `Object` ↦ `EClass` creates the transformation of all MetaEdit objects (instances of a certain object type) to EObjects (instances of the EClass corresponding to the object type). Another example is the rule `Relationship` ↦ `EClass` which creates the transformation of all MetaEdit relationships (instances of a certain relationship type) to EObjects (instances of the EClass corresponding to the relationship typ).

Moreover, model data such as symbol position or diagram name is transformed into EMF models. Hence, the abstract EMF metamodel contains EClasses such as 'Symbol' or 'Diagram'.

5. Implementation of the Bridge

We have prototypically realized the bridge in Java as an Eclipse plug-in. This bridge mainly consists of three parts. The first part is the M2-level transformation which creates EMF metamodels from MetaEdit language definitions. MetaEdit provides an XML export for the language definition. The schema of these XML files is vendor-specific and described in the tool documentation[1]. We have implemented a GOPRR-Reader which can query the XML export and return, for instance, all graph types, object types, and relationship types. Furthermore, we have implemented the transformation rules in Java by using the EMF API. The implementation builds an in-memory object model and serializes the objects as an XMI file in Ecore format.

The second part of the bridge is the M1-level transformation which transforms MetaEdit models to EMF models. MetaEdit provides two possibilities to access model data, (1) an XML format being similar to the metamodel XML format, (2) a Simple Object Access Protocol (SOAP) API. We use the SOAP API because it provides methods such as objectSet() or relationshipSet() in order to query model elements, and methods such as type() to get the corresponding metamodel elements. The transformation navigates through the MetaEdit model and creates EMF model elements, whereby the EMF model element is an instance of the EMF metamodel element corresponding to the MetaEdit metamodel element. Analogous to the M2-level transformation, an EMF model is created in a dynamic way (i.e. in-memory) and is serialized as an XMI file.

The third part of the bridge is the M1-level transformation which transforms EMF models to MetaEdit models. The transformation also uses the EMF API to navigate through the EMF models and creates a corresponding MetaEdit model element for each EMF model element by using the SOAP API.

6. Application of the Bridge

A concrete example will allow to demonstrate how we can reuse validation rules of Event-Driven Process Chains (EPC) (Nüttgens and Rump 2002) which were already implemented as part of a validation approach for business process models (Kühne et al. 2008).

EPC is a graphical modeling language to describe business processes. EPC models consist of nodes and arcs. Nodes can be functions (activities which need to be executed, depicted as rounded boxes), or events (representing pre- and

[1] http://www.metacase.com/support/45/manuals/mwb/Mw.html

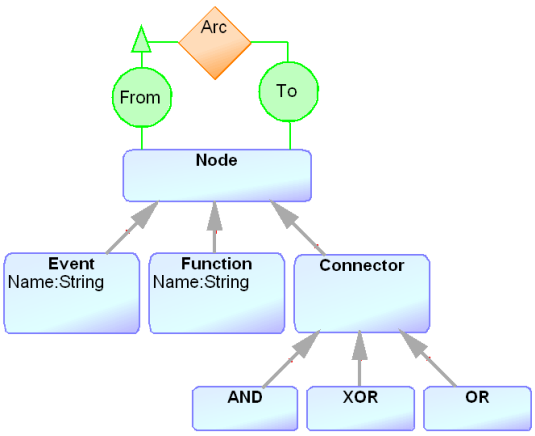

Figure 6. EPC Metamodel in MetaEdit-specific Notation.

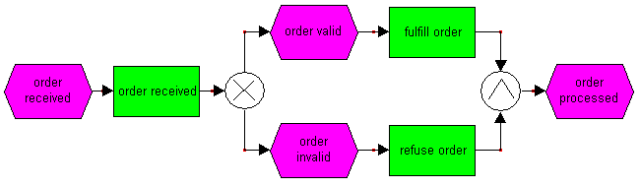

Figure 7. Incorrect EPC Model in MetaEdit.

postconditions of functions, depicted as hexagons), or connectors. Arcs between these elements represent the control flow. Connectors are used to model parallel (AND-connectors ⓐ) and alternative (XOR-connector ⓧ and OR-connector ⓥ) executions. Figure 6 shows the corresponding EPC metamodel in a MetaEdit-specific notation. There are the following objects types: 'Node', 'Event', 'Function', 'Connector', 'AND', 'XOR' and 'OR'. The arc concept is realized by a relationship type 'Arc' which is connected with 'Node' by the role types: 'From' and 'To'. Furthermore, the types 'Event' and 'Function' have a property type 'Name'.

After describing the EPC language definition in MetaEdit, we can create EPC models which are to be checked later. Figure 7 shows such an EPC model which is incorrect because of the mismatch between the type of the first connector and the type of the second connector. This construct will always result in a deadlock because the XOR-split starts only one control flow and the AND-join waits for both flows to be completed.

Now, we can export the metamodel and model from MetaEdit to Eclipse EMF by using the bridge. Afterwards, we need to transform the exported EMF-MetaEdit models to EMF models conforming to an EPC metamodel used by the validation rules. This transformation is easy to realize by a model-to-model transformation in Eclipse EMF.

Thereafter, we can apply the validation rules to check the models. The rules are expressed in the Check language from openArchitectureWare. Listing 1 shows an example rule. A rule is introduced by a context and an optional if-clause specifying a set of model elements that should be validated. The ERROR keyword and the following message signal a violated validation rule. The boolean expression after the colon provides a validation assertion which holds for valid models. In the given case, the rule in Listing 1 detects the mismatch of the connectors from Figure 7 and creates an error message. Currently, the error messages are displayed in the Eclipse workbench but it is also possible to create error messages in the MetaEdit workbench by using the SOAP API.

```
// XOR−AND−Mismatch
context epc::Connector if (this.isAndJoin())
    ERROR "Mismatched XOR−split ..."
    this.allPredessesors().notExists(p| p.isXorSplit()
     && p.seseMatch(this));
```

Listing 1. XCheck Rule: Mismatched XOR-Split/AND-Join

The above example is very specific but it is also possible to use other EMF tools. Furthermore, the usage of other bridges from Eclipse EMF to other metamodel hierarchies enables the usage of further tools. For instance, the Eclipse EMF to Web Ontology Language bridge, implemented in the EODM project[2], enables the application of typical ontologies tools such as reasoner.

7. Summary and Conclusion

In this paper, we developed an interface for the exchange of metamodels and models between MetaEdit and Eclipse EMF by applying the concept of M3-Level-Based Bridges. For this purpose, we explored the MetaEdit language definition concepts and the underlying repository structure. Furthermore, we described the metametamodel Ecore and the generic model storage structure of Eclipse EMF. Based on this information, we specified a M3-level mapping and derived a M2-level and M1-level transformation implemented as an Eclipse plug-in. After, we demonstrated the bridge by an example of EPC validation. The example showed the modeling of EPCs and the application of validation in Check.

M3-Level-Based Bridges have already been used repeatedly to achieve (meta)model interchange between different tools based on metamodel hierarchies. As a result of the development and the study of this bridge and other bridges, we can say that this approach is useful to build tool chains and to re-use models and model operations. But we also identified the following typical problems that need further research:

Expressive power Often, the metametamodels are different, i.e., they provide different metamodeling concepts to express metamodels. For instance, some metametamodels support the `Model type` concept, others do not support this concept natively. Hence, the mapping between metametamodels can be very complex or even impossible.

Synchronization The synchronization of models and metamodels during the exchange is important in real-world use cases. Therefore, we need approaches for traceability links, model differences and model merging.

In the future work, we want to solve the problems mentioned above. Moreover, we want to develop further M3-Level-Based Bridges and want to explicit our experience in terms of a guide which helps with the development of bridges.

[2] http://www.eclipse.org/modeling/mdt/

References

The AMMA Platform. http://www.sciences.univ-nantes.fr/lina/atl/AMMAROOT/, 2008.

Eclipse Modeling Framework Technology (EMFT). http://www.eclipse.org/modeling/emft/, 2008.

Eclipse Graphical Modeling Framework. http://www.eclipse.org/gmf/, 2008.

openArchitectureWare. http://www.openarchitectureware.org/, 2008.

Colin Atkinson and Thomas Kühne. Model-Driven Development: A Metamodeling Foundation. *IEEE Software*, 20(5):36–41, September 2003.

Frank Budinsky, David Steinberg, Ed Merks, Ray Ellersick, and Timothy J. Grose. *Eclipse Modeling Framework*. The Eclipse Series. Addison Wesley, 2004. ISBN 01311425420.

Jean Bézivin, Hugo Brunelière, Frédéric Jouault, and Ivan Kurtev. Model Engineering Support for Tool Interoperability. In *Proceedings of the 4th Workshop in Software Model Engineering (WiSME)*, 2005a.

Jean Bézivin, Christian Brunette, Régis Chevrel, Frédéric Jouault, and Ivan 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, 2005b.

Jean Bézivin, Guillaume Hillairet, Frédéric Jouault, Ivan Kurtev, and William 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, 2005c.

Steve Cook, Gareth Jones, Stuart Kent, and Alan Cameron Wills. *Domain Specific Development with Visual Studio DSL Tools (Microsoft .Net Development)*. Addison-Wesley Longman, 2007. ISBN 0321398203.

Keith Duddy, Anna Gerber, and Kerry Raymond. Eclipse Modelling Framework (EMF) import/export from MOF / JMI. Technical report, CRC for Enterprise Distributed Systems Technology (DSTC), 2003.

Marcos Didonet Del Fabro, Jean Bézivin, Frédéric Jouault, Erwan Breton, and Guillaume Gueltas. AMW: A Generic Model Weaver. In *Proceedings of the 1ère Journée sur l'Ingénierie Dirigée par les Modèles (IDM05)*, 2005.

Ralf Gitzel and Tobias Hildenbrand. A Taxonomy of Metamodel Hierarchies. *Working papers / Lehrstuhl für ABWL und Wirtschaftsinformatik*, 2005.

John Grundy, John Hosking, Jun Huh, and Karen Li. Marama: an Eclipse meta-toolset for generating multi-view environments. *IEEE/ACM International Conference on Software Engineering*, 2008.

Frédéric Jouault and Ivan Kurtev. Transforming Models with ATL. In *Proceedings of the Model Transformations in Practice Workshop at MoDELS 2005*, Montego Bay, Jamaica, 2005.

Gabor Karsai, Andras Lang, and Sandeep Neema. Tool Integration Patterns. *Workshop on Tool Integration in System Development (TIS)*, 2003.

Steven Kelly. *Towards a Comprehensive MetaCASE and CAME Environment: Conceptual, Architectural, Functional and Usability Advances in MetaEdit+*. PhD thesis, University of Jyväskylä, 1997.

Steven Kelly and Juha-Pekka Tolvanen. *Domain-Specific Modeling: Enabling Full Code Generation*. John Wiley & Son, Inc., 2008. ISBN 0470036664.

Heiko Kern and Stefan Kühne. Model Interchange between ARIS and Eclipse EMF. In Juha-Pekka Tolvanen, Jeff Gray, Matti Rossi, and Jonathan Sprinkle, editors, *7th OOPSLA Workshop on Domain-Specific Modeling at OOPSLA 2007*, 2007.

Stefan Kühne, Heiko Kern, Volker Gruhn, and Ralf Laue. Business Process Modelling with Continuous Validation. In Cesare Pautasso and Jana Köhler, editors, *1st International Workshop on Model-Driven Engineering for Business Process Management*, 2008.

Thomas Kühne. Matters of (Meta-) Modeling. *Software and Systems Modeling*, 5(4):369–385, 2006. doi: 10.1007/s10270-006-0017-9.

Michael Lawley and Jim Steel. Practical Declarative Model Transformation With Tefkat. *Satellite Events at the MoDELS 2005 Conference*, LNCS Vol. 3844, 2005.

Akos Ledeczi, Miklos Maroti, Arpad Bakay, Gabor Karsai, Jason Garrett, Charles Thomason, Greg Nordstrom, Jonathan Sprinkle, and Peter Volgyesi. The Generic Modeling Environment. In *IEEE International Workshop on Intelligent Signal Processing*, 2001.

MetaCase. MetaEdit+. http://www.metacase.com/, 2008.

Markus Nüttgens and Frank J. Rump. Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In *Promise 2002 - Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen*, pages 64–77, 2002.

Meta Object Facility (MOF) Specification 1.4. Object Management Group, 2002. URL `http://www.omg.org/docs/omg/03-06-01.pdf`.

Juha-Pekka Tolvanen. *Incremental Method Engineering with Modeling Tools: Theoretical Principles and Empirical Evidence*. PhD thesis, University of Jyväskylä, 1998.