Integration of Microsoft Visio and Eclipse Modeling Framework Using M3-Level-Based Bridges

Heiko Kern and Stefan Kühne

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

Abstract. Nowadays there are powerful tools for Model-Driven Development. An ongoing problem is the insufficient tool interoperability which complicates the development of complete tool chains or the reuse 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 Microsoft Visio with strengths in modelling and information visualization and the second tool is the Eclipse Modeling Framework with advantages in model processing by transformation and generation tools.

1 Introduction

The approach of Model-Driven Development (MDD) [1] has gained in relevance during the last decade. The consequent use of models, modelling languages and model operations as central concepts in this approach leads to several advantages: a better handling of complexity, more efficient development processes, and improved software quality, to name a few. Today powerful tools are available for the support of typical MDD tasks such as language definition, modelling or model processing. But an ongoing problem is the insufficient tool interoperability [2, 3]. This lack of interoperability complicates the development of complete tool chains or the reuse of existing metamodels, models, and model operations in a development process.

In this article, we present the approach of M3-Level-Based Bridges (M3B) to achieve tool interoperability between different MDD tools. We apply this approach to realise the interchange of metamodels and models between two selected tools: Microsoft Visio [4] and Eclipse Modeling Framework (EMF) [5]. Microsoft Visio is a widely used commercial tool that is mainly suitable for modelling and visualising information. Visio provides the definition of modelling languages by using stencils. The tool is part of the Microsoft Office Suite that enables the integration into other Microsoft products. In addition, there are many tools and add-ons (ViFlow[1], SemTalk[2], etc.) that provide the modelling in specific domains, for instance, in business process management. The second tool or, more

[1] http://www.viflow.com
[2] http://www.semtalk.com

precisely, framework is given by the Eclipse Modeling Framework and by tools (Epsilon[3], openArchitectureWare[4], etc.) based on this framework. EMF allows the definition of domain-specific metamodels and provides basic functionalities, such as a serialization function, a generator language, and an API to access (meta)models. In summary, the motivation to build an M3B between Visio and EMF is that both tool spaces can benefit from the created interoperability because Visio has advantages in modelling and EMF provides a wide range of model processing tools.

The paper is structured as follows: in the following section, we will give a conceptual overview of M3B. As the bridge is based on the mapping between metametamodels, we will explore the metamodel hierarchy of Visio and EMF in section 3. Then, we will present the development of the bridge and show an example in section 4. Lastly, we will give an overview of related works in section 5 and will present our conclusions in section 6.

2 M3-Level-Based Bridges

The M3B approach is well-established and has been successfully applied in building bridges between different tool spaces. M3Bs are, for instance, the ARIS[5]-to-EMF bridge [6], MetaEdit[6]-to-EMF bridge [7], or XML-to-EMF [5] bridge. Although the implementation of such bridges can differ in technical terms, the conceptual approach is the same.

A basic prerequisite to construct M3Bs is the existence of a metamodel hierarchy [8] consisting of three levels. At the lowest level (M1-level) are models which describe a software system. The structure of these models and the available concepts that can be instantiated in these 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. Models, metamodels and the metametamodel are connected by a linguistic instance relationship [8] that can also be denoted as conform-to relationship [9].

Based on the existence of such hierarchies, the basic step to build an M3B is the mapping between the metametamodels. The mapping consists of different mapping rules that specifiy the relation between semantically equivalent concepts. Semantic equivalence means, for instance, that the concepts at M3-level which express relationships at M2-level are mapped onto each other. Further semantic concepts are described in [7] and include, for instance, object type, attribute type or data type. 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 conform-to relationship is realised between each level. The M2-level transformation maps metamodels between hierarchies. The

[3] http://www.eclipse.org/gmt/epsilon/
[4] http://www.openarchitectureware.org
[5] http://www.ids-scheer.com
[6] http://www.metacase.com

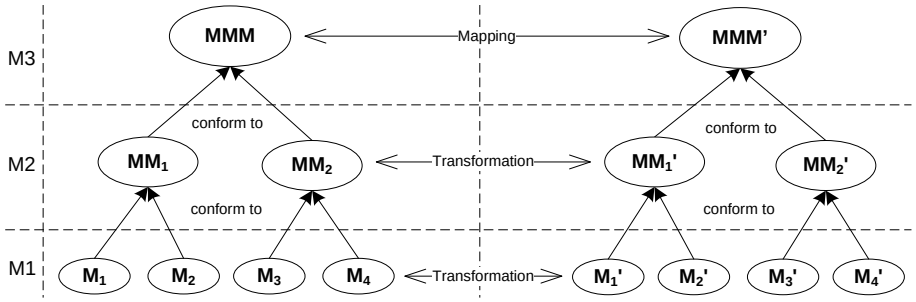

Fig. 1. Pattern of M3-Level-Based Bridges

metamodels in both hierarchies are isomorphic. Analogous to the M2-level transformation, the M1-level transformation enables the mapping of models. These models are also isomorphic. Figure 1 shows the concept of M3B.

The use of the transformation taxonomy from Mens et al. [10] characterise the transformations as follows. The M2-level transformation is horizontal and exogenous because the abstraction level stays the same and the metametamodels are different. The M1-level transformation is horizontal and endogenous because the abstraction level also stays the same and the metamodels are equal.

3 Modelling with Visio and EMF

3.1 Model Structure in Visio

A linguistic metamodel hierarchy consisting of three levels occurs in Visio: V3 model at M3-level, stencils at M2-level, and Visio models at M1-level. In this section we present the data structure for Visio models at M1-level in order to describe the M1-level transformation later.

To analyse the data structure we have used mainly the Visio object model[7] (Visio's programming interface) and the user interface. A snippet of the model structure is shown in Figure 2. Visio models are stored in documents that are usually files with a name. A document consists of pages. A page has also a name and is a canvas that contains shape elements. A shape is a model element and is an instance of a master from a stencil. Hence, a shape has a reference attribute to a master. A shape can have its own data properties (defined only at M1-level) and data properties defined in a master (defined at M2-level). A page can likewise have data properties. Data properties store values with a specified data type. Visio supports data types such as text, numbers, time, date, currency and list. Furthermore, a shape can contain other shapes and can be connected with other shapes.

[7] http://msdn.microsoft.com/en-us/library/aa342179.aspx

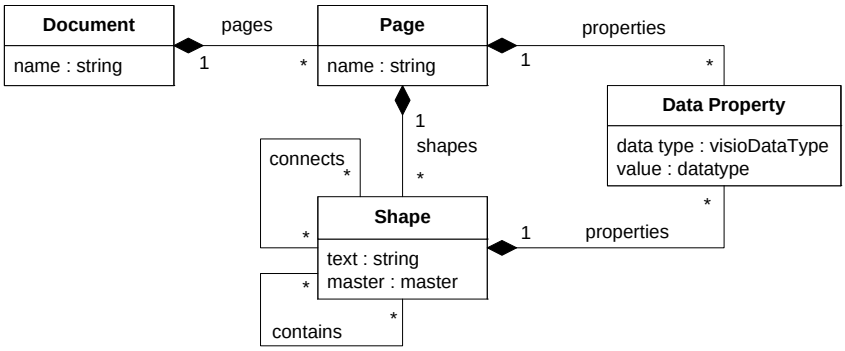

Fig. 2. Data Structure of Visio Models

3.2 Metamodelling in Visio

Visio offers the functionality for building Domain-Specific Languages (DSL) [11]. The (concrete and abstract) syntax of a DSL can be defined by stencils. Analogous to the analysis of the model structure, we have studied the object model and the user interface. The result of the analysis is a metametamodel that explicates the language definition concepts. This metametamodel is here denoted as V3 model (see Fig. 3).

Contrary to other metametamodels, the V3 model is relatively simple. It consists of `Stencil`, `Master`, and `Data Property`. A stencil (instance of `Stencil`) is a set of masters (instances of `Master`). A master describes a class of model elements (in Visio denoted as shape – see previous subsection) which can exist on their own. Masters can occur in two forms: a simple shape and a connection shape. In the context of a graph, a simple shape and a connection shape can be regarded as a node and an edge, respectively. A further metamodelling concept is to define properties (instances of `Data Property`). Each data property is assigned to a master, has a data type, and can have a default value.

In addition to defining an abstract syntax, Visio provides the description of a concrete syntax. Each master owns a graphical form which can be any kind of graphical construct (e.g. rectangle, circle, line, icon, star, polygon, ellipse, a combination of these figures, etc.). Data properties can also be displayed as textual annotation.

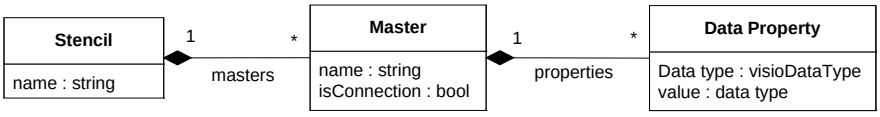

Fig. 3. Definition of Visio Stencils

3.3 Metamodels and Models in EMF

Contrary to Visio, developed especially for modelling and information visualization, EMF is designed to support the development of (Eclipse) applications. Models describing EMF applications can be regarded as metamodels and instances of these metamodels can be regarded as models. All metamodels are instances of the metametamodel Ecore which is implemented in EMF.

A simplified subset of Ecore is shown in Figure 4. The main elements of Ecore are `EPackage`, `EClass`, `EReference` and `EAttribute`. An EPackage is identified by an Uniform Resource Identifier and contains a set of EClassifiers. An EClassifier can be an EClass or an EDataType. An EClass defines an EMF metamodel element that represents a set of similar model entities. EClasses can have EReferences which express unidirectional relationships between two EClasses. An EClass can additionally have EAttributes to express properties of this EClass. The range of the attribute values are specified by an EDataType such as EInt, EString or EDate. A further metamodelling concept is the inheritance between EClasses.

Apart from considering the structure of Visio models (see Section 3.1), we need to know the structure of EMF models in order to build the M1-level transformation. Every model element in 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.

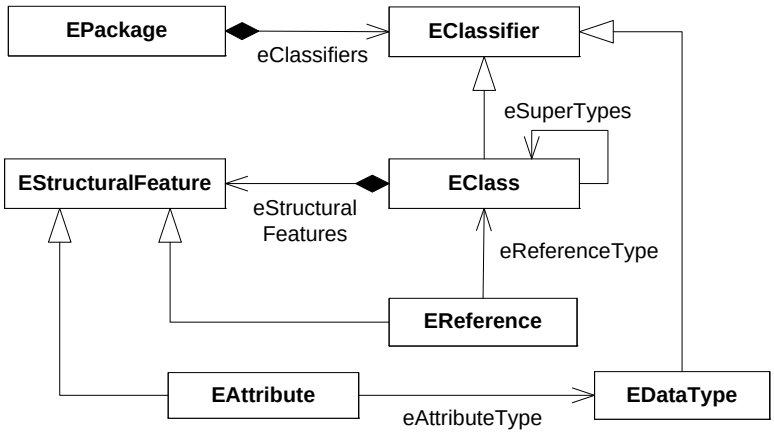

Fig. 4. Subset of Ecore

4 Microsoft Visio to EMF Bridge

4.1 Overview

Microsoft Visio and Eclipse EMF can be divided into three levels: M3 (V3 and Ecore), M2 (Visio stencils and EMF metamodels), and M1 (Visio models and EMF models). Based on this level structure, we apply the approach of M3-Level-Based Bridge (see Fig. 5). The M3-level mapping specifies an unidirectional mapping from V3 to Ecore. Using this mapping, we can derive the transformation rules at M2-level which export stencils to EMF. Several transformation rules map different Visio concepts onto one Ecore concept. To distinguish the different Visio concepts in Ecore, we introduce an abstract EMF metamodel that approximates the data model structure of Visio (see Fig. 6). All exported metamodel elements are inherited from a corresponding abstract metamodel element. For our purpose, the export of stencils is sufficient and we will therefore ignore the import of EMF metamodels to Visio. The consideration of this import would 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 Visio models.

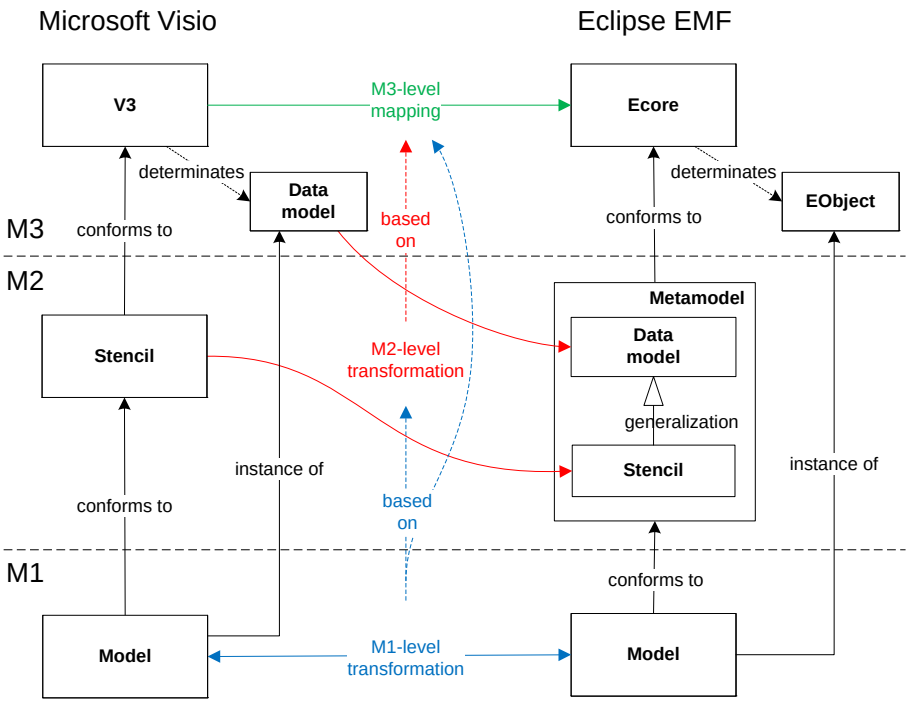

Fig. 5. Overview of the Visio-to-EMF Bridge

4.2 M3-Level Mapping

We propose the following mappings:

`Stencil` ↦ `EPackage`**:** In Visio a stencil contains a set of modelling concepts in terms of masters. A similar concept in Ecore is the `EPackage` concept. `EPackage` can also group modelling concepts. Therefore, we map `Stencil` onto `EPackage`. The `name` of `Stencil` maps onto the `name` of `EPackage` and the association `masters` maps onto the reference `eClassifiers` from `EPackage`.

`Master` ↦ `EClass`**:** In Visio a master can define a set of model elements. The corresponding concept in Ecore is the `EClass` concept which can also define a set of model elements. Hence, we map `Master` onto `EClass`. The stencil of a master is equivalent to the EPackage of the corresponding EClass. The `name` of `Master` maps onto the `name` of `EClass`.

`Data Property` ↦ `EAttribute`**:** A data property in Visio and an EAttribute in Ecore can be used to describe properties of a set of modelling elements. Thus, we map `Data Property` onto `EAttribute`. The name of a data property is equal to a name of an EAttribute. Visio supports data types such as: string, number, list, boolean, currency, date and time. We map these types as follows {Visio data type ↦ EMF data type}: {string, currency, date, and time ↦ EString}, {number ↦ EInt}, {booelan ↦ EBoolean}, {list ↦ multi-value EAttribute}.

4.3 M2-Level Transformation

The M2-level transformation consists of transformation statements which are derived from the M3-level mapping specification. The mapping rule `Stencil` ↦ `EPackage` transforms all stencils in EPackages. The rule `Master` ↦ `EClass` transforms all masters in EClasses. If the masters are a simple shape, then the corresponding EClasses are inherited from the abstract EClass "Shape" (see Fig. 6). If the masters are a connection shape, then the corresponding EClasses are inherited from the abstract EClass "Connection" (see Fig. 6). The last rule `Data Property` ↦ `EAttribute` transforms all data properties to EAttributes with the equivalent data type and optional with a default value.

4.4 M1-Level Transformation

The M1-level transformation contains transformation statements which are also derived from the M3-level mapping specification. The mapping rule `Master` ↦ `EClass` transforms all shapes (instances of a certain master) to EObjects (instances of the EClass corresponding to the master). The rule `Data Property` ↦ `EAttribute` causes the transformation of all property values of a shape into values of an EAttribute which corresponds to the shape's EClass. The rule `Stencil` ↦ `EPackage` has no effect at the M1-level. In addition to the derived transformation rules, graphical model data such as symbol position or symbol

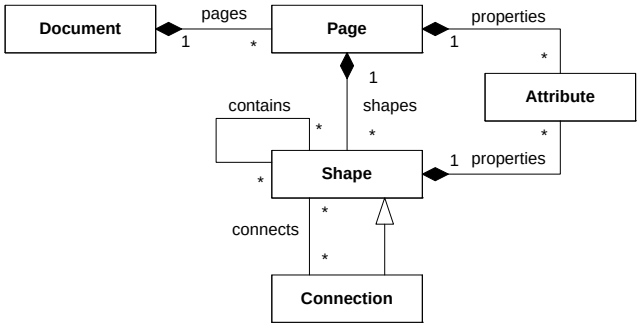

Fig. 6. Abstract Visio Data Model in EMF

size is transformed into EMF model data. Furthermore, all data properties defined at M1-level are transformed into instances of the EClass "Attribute" taken from the abstract metamodel.

4.5 Implementation

We have prototypically realised the Visio-to-EMF bridge as an Eclipse plug-in. The bridge consists mainly of three parts. The first part is the M2-level transformation which creates EMF metamodels from stencils. We have written a C# program that exports stencil data as an XML document. This document is read by a stencil reader that is implemented in Java. The model is transformed afterwards, according to the M2-level rules. The transformation builds an in-memory object model and serialises the objects as an XMI file in Ecore format.

The second part of the bridge is the M1-level transformation with Visio models as input and EMF models as output. Analogous to the stencil export, a C# program creates an XML document containing the Visio model data. Furthermore, we have implemented a model reader and the transformation rules in Java. The reader navigates through the Visio model and creates EMF model elements. The EMF model is also created in a dynamic way (i.e. in-memory) and is serialised as an XMI file.

The third part of the bridge is the M1-level transformation with EMF models as input and Visio models as output. The transformation creates an XML document from the EObject model with XPand[8]. The document contains the Visio model data which is parsed by a C# program. This program creates the Visio model by using the Visio programming interface.

4.6 Use Case

A concrete example should illustrate the Visio-to-EMF bridge. In this case we want to export Event-Driven Process Chains (EPC) [12] from Visio to EMF in

[8] XPand is a part of openArchitecureWare.

order to reuse a validation approach for business processes [13] that we have already implemented with EMF-based tools.

EPC is a graphical modelling language to describe business processes. EPC models consist of nodes and arcs. Nodes can be functions, events, 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 7 shows at the left-hand side the metamodel hierarchy with the V3 model, EPC stencil and EPC model that is to be transformed into EMF. The results of the transformations are the EMF metamodel and the EMF model at the right-hand side. Afterwards, we need to transform the exported EMF-Visio models into EMF models conforming to an EPC metamodel used by the validation rules. This transformation is easy to realise by a model-to-model transformation with EMF-based transformation tools. Thereafter, we can apply the validation rules to check the models. In this case the model contains no errors.

The above example is very specific but, it is also possible to use other stencils and Visio models. Furthermore, the usage of other bridges from EMF to other metamodel hierarchies enables the usage of further tools. For instance, the ARIS-to-EMF bridge enables the application of ARIS tools.

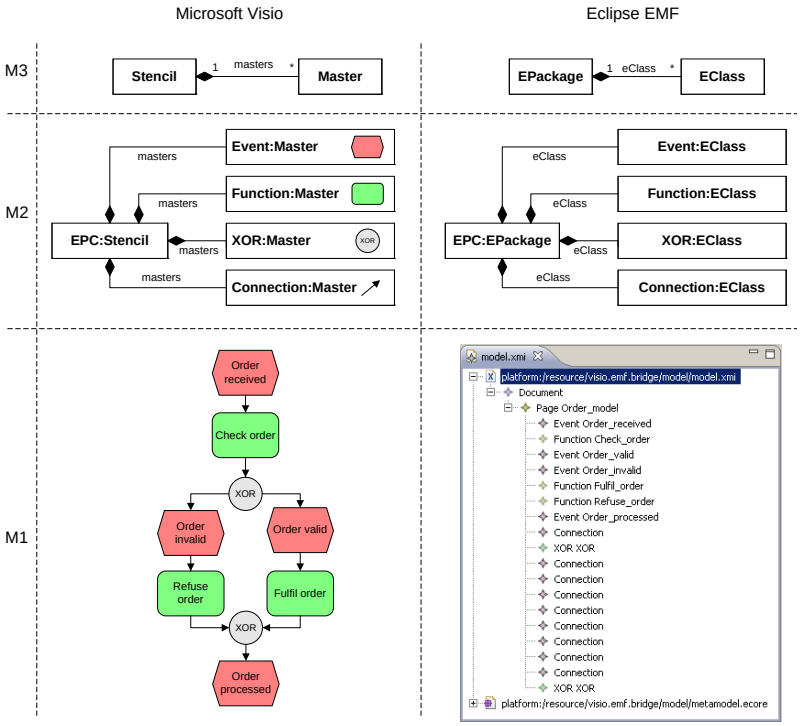

Fig. 7. Export of Visio-EPC models to EMF-EPC models

5 Related Work

As already mentioned in section 2, the M3B approach occurs in many cases such as ARIS-to-EMF [6], MetaEdit-to-EMF [7], EMF-to-XML [5] or XML-to-Relational Database [14]. All these bridges apply the same pattern to integrate different metamodel hierarchies. They define a mapping on the M3-level and use the instance-of relationship between the underlying level to transform metamodels at M2-level and models at M1-level.

In the context of tool integration, the existing approaches are manifold. Regarding the level of transformation, the M3B approach is based upon a mapping onto the M3-level. Other approaches [15, 2] work on M2-level, that is, the integration is based on a transformation between metamodels correlating to the data models of the tools.

Furthermore, there are generic data formats such as the XML Metadata Interchange [16] or the Graph eXchange Language (GXL) [17]. GXL, for instance, offers support for exchanging instance graphs together with their appropriate schema information in an uniform format. GXL is based on typed, attributed, ordered directed graphs, which are extended by concepts to support the representation of hypergraphs and hierarchical graphs. Theoretically, GXL could be used in order to realise the exchange of the data between each level. For this, each model has to be transformed into this graph format.

6 Summary and Conclusion

In this paper, we have developed an interface for the exchange of metamodels and models between Microsoft Visio and Eclipse EMF by applying the concept of M3-Level-Based Bridges. For this purpose, we explored the Visio stencil definition concepts and the underlying data structure of models. Furthermore, we described the metametamodel Ecore and the generic model storage structure of EMF. Based on this information, we specified an M3-level mapping and derived an M2-level and M1-level transformation. Based on the transformation specification, we have implemented the bridge as an Eclipse plug-in. Furthermore, we have demonstrated the bridge by an example of EPC export. 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 reuse models and model operations. In a next step, we want to describe the concept in a more formal way than presented in this paper.

References

1. Stahl, T., Völter, M.: Model-Driven Software Development – Technology, Engineering, Management. John Wiley & Sons Inc. (2006)
2. Bezivin, J., Brunelière, H., Jouault, F., Kurtev, I.: Model Engineering Support for Tool Interoperability. Proceedings of the 4th Workshop in Software Model Engineering (WiSME) (2005)
3. Karsai, G., Lang, A., Neema, S.: Tool Integration Patterns. Workshop on Tool Integration in System Development (TIS) (2003)
4. Biafore, B.: Visio 2007 Bible. John Wiley & Sons Inc. (2007)
5. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: Eclipse Modeling Framework. The Eclipse Series. Addison-Wesley Longman (2009)
6. Kern, H., Kühne, S.: Model Interchange between ARIS and Eclipse EMF. In Tolvanen, J.P., Gray, J., Rossi, M., Sprinkle, J., eds.: 7th OOPSLA Workshop on Domain-Specific Modeling at OOPSLA 2007. (2007)
7. Kern, H.: The Interchange of (Meta)Models between MetaEdit+ and Eclipse EMF Using M3-Level-Based Bridges. In Tolvanen, J.P., Gray, J., Rossi, M., Sprinkle, J., eds.: 8th OOPSLA Workshop on Domain-Specific Modeling at OOPSLA 2008. (2008)
8. Kühne, T.: Matters of (Meta-) Modeling. Software and Systems Modeling **5**(4) (2006) 369–385
9. Bézivin, J.: On the Unification Power of Models. Software and System Modeling **4**(2) (2005) 171–188
10. Mens, T., Czarnecki, K., Gorp, P.V.: A Taxonomy of Model Transformations [online]. In Bezivin, J., Heckel, R., eds.: Language Engineering for Model-Driven Software Development. Number 04101 in Dagstuhl Seminar Proceedings, Internationales Begegnungs- und Forschungszentrum (IBFI), Schloss Dagstuhl, Germany (2005)
11. Kelly, S., Tolvanen, J.P.: Domain-Specific Modeling – Enabling Full Code Generation. John Wiley & Son, Inc. (2008)
12. Keller, G., Nüttgens, M., Scheer, A.W.: Semantische Prozeßmodellierung auf der Grundlage Ereignisgesteuerter Prozeßketten (EPK) (in German). Technical Report Heft 89, Institut für Wirtschaftsinformatik, Universität des Saarlandes, Saarbrücken (1992)
13. Kühne, S., Kern, H., Gruhn, V., Laue, R.: Business Process Modelling with Continuous Validation. In Pautasso, C., Köhler, J., eds.: 1st International Workshop on Model-Driven Engineering for Business Process Management. (2008)
14. Balmin, A., Papakonstantinou, Y.: Storing and querying XML data using denormalized relational databases. The VLDB Journal–The International Journal on Very Large Data Bases **14**(1) (2005) 30–49
15. Amelunxen, C., Klar, F., Königs, A., Rötschke, T., Schürr, A.: Metamodel-based Tool Integration with MOFLON. Proceedings of the 30th International Conference on Software Engineering (2008) 807–810
16. Object Management Group: MOF 2.0/XMI Mapping, Version 2.1.1. Technical report (2007)
17. Winter, A., Kullbach, B., Riediger, V.: An Overview of the GXL Graph Exchange Language. Volume LNCS 2269., Springer (2002) 324–336