Tool Integration by Model Transformations based on the Eclipse Modeling Framework

Karsten Ehrig* and Gabriele Taentzer and Dániel Varró*****
*University of Leicester, United Kingdom
**Technical University of Berlin, Germany
***Budapest University of Technology and Economics, Hungary

Abstract. *In the paper, we propose various approaches for tool integration based on model transformations over the Eclipse Modeling Framework (EMF). EMF is a key technology for tool integration, which provides a framework for developing domain-specific modeling languages by automatically generating Java code for model manipulation. Model transformations can be captured by graph transformation systems, which support visual specifications based on rules and patterns. Three levels of tool integration are identified: (i)* model-level integration *carries out model transformations in existing transformation tools by importing and exporting EMF models, (ii)* interpreted EMF transformations *take an EMF model of the transformation system, and manipulate EMF models according to the system by calling EMF interfaces, finally (iii)* compiled transformer plug-ins *generate stand-alone transformer programs in Java which are responsible for model manipulation.*

Keywords: model transformation, tool integration, graph transformation, Eclipse

1 Introduction

In model-driven software development, the Eclipse Modeling Framework (EMF) [EMF06] is becoming a key technology for tool integration. It is a framework for developing domain-specific modeling languages defined by an appropriate class diagram (metamodel) by automatically generating Java code which supports to create, modify, store, and load instances of the model. Moreover, it provides generators to support the basic editing of EMF models. EMF unifies three important technologies: Java, XML, and UML. Regardless of which one is used to define a model, an EMF model can be considered as the common representation that subsumes the others.

Graph transformation [EEPT06] provides a rule and pattern-based specification technique to manipulate graph-based models. Graph transformation approaches and tools frequently serve as an underlying framework for capturing model transformations with and between modeling languages at various levels of abstraction.

Model transformations have been identified as a key challenge in the SENSORIA European IST project.

The aim of SENSORIA is to develop a novel comprehensive approach to the engineering of software systems for service-oriented overlay computers where foundational theories, techniques and methods are fully integrated in a pragmatic software engineering approach. It focuses on global services that are context adaptive, personalisable, and may require hard and soft constraints on resources and performance, and plans to take into account the fact that services have to be deployed on different, possibly interoperating, global computers, to provide novel and reusable service-oriented overlay computers.

Model transformations aim to serve as a bridge between various SENSORIA tools used for the design, the discovery, the verification, the implementation or the deployment of services.

In the current paper, we propose to use EMF as general framework for integrating tools by model transformations captured by graph transformation techniques in order to integrate the advantages of both techniques. The envisaged tool integration can be carried out in three different ways:

1. **Model-level integration.** A first (traditional) approach (Sec. 3) is to use existing model transformation tools by providing appropriate facilities to import and export for EMF models. Starting from an EMF source model, it is first imported into the designated model transformation tool. Then the actual model transformation is carried out using the internal model representation of the transformation tool. Finally, the result of the transformation is exported into a target EMF model.

2. **Interpreted transformations of EMF models.** A second approach (Sec. 4) is to store model transformations also in the form of EMF models and write an interpreter, which takes a source model and transformation rules in its EMF representation as input, and generates the designated target model as output. In this solution, the transformation engine is a general one, which should be capable of transforming any models corresponding to EMF.

3. **Compiled transformer plugins in Java.** A third approach (Sec. 5) is to take a specification of a model transformation and compile it into a stand-alone transformer plugin (i.e. a Java program), which takes a source EMF instance as input and generates a target EMF instance as output.

As model transformation is becoming an engineering discipline (*transware* [VP04]), conceptual and tool support is needed for the entire life-cycle, i.e. the specification, design, execution, validation and maintenance of transformations. However, different phases of transformation design frequently set up conflicting requirements, and it is difficult to find the best compromise. For instance, the main driver in the execution phase is performance, therefore, a *compiled MT approach* (where a transformation is compiled directly into native source code) is advantageous. On the other hand, *interpreted MT approaches* (where transformations are available as models) have a clear advantage during the validation (e.g. by interactive simulation) or the maintenance phase due to their flexibility.

After a brief summary on EMF (2), Section 3 first details model-level integration techniques. Then Sec. 4 proposes the use of interpreted EMF transformations for tool integration. In Sec. 5, we outline how automatically generated stand-alone transformer plug-ins can serve as a transformation based integration technique. Finally, Secs. 6 and 7 conclude this paper.

2 Eclipse Modeling Framework (EMF)

The Eclipse Modeling Framework (EMF) [EMF06] provides a modeling and code generation framework for Eclipse applications based on structured data models. The modeling approach is similar to that of MOF, actually EMF supports Essential MOF (EMOF) as part of the OMG MOF 2.0 specification [EMO06]. The type information of sets of instance models is defined in a so-called core model corresponding to the metamodel in EMOF. The core (or metamodel for core models) is the Ecore model. It contains the model elements which are available for EMF core models in principle.

From an EMF model, a set of Java classes for the model and a basic, tree based editor can be generated. The generated classes provide basic support for *creating/deleting* model elements, persistency operations like *loading and saving*, and a *notification mechanism* to support the model-view-controller paradigm by sending events on model changes.

Relations between EMF model classes are handled by special EMF lists, extending the Java list classes. Moreover, EMF models can be used as underlying models in new application plugins. But in many cases, the EMF model by its own is not powerful enough to express the complete model behavior. Therefore the generated code can be extended by the developer in order to add new functionalities that are not expressed in the EMF model.

The practical relevance of EMF is clearly demonstrated by the fact that many software vendors (like IBM and Borland) have recently chosen to commonly use EMF models in their products to facilitate easy tool integration.

3 Model-Level Integration

The first approach (illustrated in Fig. 1) for integrating model transformations is to use the import / export functionalities of an existing model transformation tool like VIATRA2 [VIA06] and development tools for ATL [ATL06].

- In the first step, the designated source EMF model is imported into the model transformation tool either by generating some tool-specific model descriptions, or calling directly the model manipulation interfaces of the transformation tool from an EMF model instance.
- Then the entire model transformation is designed and executed using the designated model transformation tool. The result of the transformation is obtained in a tool-specific format, which is again either a textual or an in-memory representation of the target model.
- Finally, the target EMF model is generated from the tool-specific target model by calling the appropriate model manipulation methods of the target EMF API or generating an XMI 2.0 document (specific to the target modeling language) which can be parsed automatically by the EMF framework.

This approach primarily provides an off-line synchronization between EMF and tool-specific models, i.e. the models are only integrated into EMF after the target model has been generated by the transformation. However, an advanced (on-the-fly) model-level integration approach may aim at the synchro-

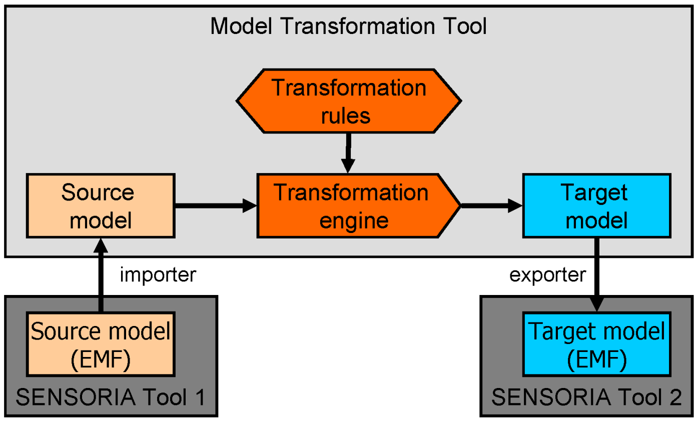

Figure 1: Model-level integration by existing model transformation tools

nization of notification mechanisms of EMF and the used model transformation framework. As a result, changes in both models are propagated immediately between the two platforms.

Consistency recovery. Since a model-level integration approach is basically independent from EMF, synctactic consistency of the generated target EMF models are typically checked only after the transformation (but not during the execution). Thus, in case of erroneous model transformations, errors are detected and recovery actions are performed late during the execution process. In any case the error checks depend on the functionalities of the used model transformation tool but most of these are typically post-transformation checks. The following approaches aim at providing an early (on-the-fly) error recovery by forcing transformations to use EMF interfaces for model manipulation. Furthermore, the transformation system has to be created within the model transformation tool, i.e. completely independent of EMF models.

4 Interpreted Transformations of EMF Models

The second approach (illustrated in Fig. 2) for performing model transformations is to take a specification of a model transformation and interpret the rules by an existing graph transformation engine like AGG [AGG06] or VIATRA2 [BV06].

In [BK06] the foundations of rule-based transformations in EMF are described and applied using the AGG graph transformation tool environment [AGG06]. Basically, an EMF transformation is a rule-based modification of an EMF source model resulting in an EMF target model. Both, the EMF source and target models are typed over EMF metamodels which itself are again typed over Ecore. The transformation rules are typed over a *transformation metamodel* (*Transformation Meta* in Fig. 2) which is independent from the used transformation tool, but somehow dependent on the transformation approach. The trans-

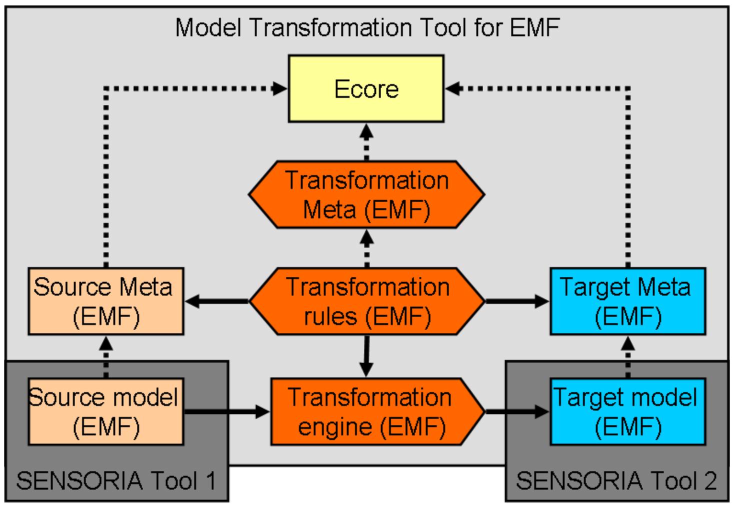
Model Transformation Tool for EMF
Ecore
Transformation Meta (EMF)
Source Meta (EMF)
Transformation rules (EMF)
Target Meta (EMF)
Source model (EMF)
Transformation engine (EMF)
Target model (EMF)
SENSORIA Tool 1
SENSORIA Tool 2

Figure 2: Interpreted model transformations for EMF

formation metamodel itself is an instance of Ecore again (see Fig. 2 where the *typed over* relations are indicated by dashed arrows). In the following, we describe the core concepts for transformation systems, inspiered by graph transformation concepts.

In general a transformation system consists of a set of transformation rules. Rules are expressed mainly by two object structures LHS and RHS, the left and right-hand sides of the rule. Furthermore, a rule has a mapping between LHS and RHS patterns. The left-hand side LHS represents the pre-conditions of a rule, while the right-hand side RHS describes its post-conditions. Those pattern parts of the LHS which are mapped to the RHS, describe a structure part which has to occur in the EMF source model, but which is not changed during the transformation. All symbols and links of the LHS not mapped to the RHS define the part which shall be deleted, and all symbols and links of the RHS which are not used for mapping, define the part to be created.

The applicability of a rule can be further restricted by additional application conditions. As already mentioned above, the LHS of a rule formulates some kind of positive condition. In certain cases also *negative application conditions* (NACs) which are pre-conditions prohibiting certain object structures, are needed. If several NACs are formulated for one rule, each of them has to be fulfilled.

Performing a transformation step which applies a rule at a selected match, the resulting object structure is constructed in two passes: (1) all objects and links present in the LHS but not in the RHS are deleted; (2) all object and links in the RHS but not in the LHS are created. A transformation, more precisely a transformation sequence, consists of zero or more transformation steps.

To apply the defined transformation rules to a given EMF model, we either select and apply the rules step-by-step, or take the whole rule set and let it apply as long as possible. A transformation step with a selected rule is defined by first finding a match of the left-hand side in the current instance model. A pattern matches to a model if its structure can be found in the model such that the types and attribute values are compatible. In general, a pattern can match different parts of a model. In this case, one of the possible matches is selected, either randomly or by the user. Certain graph transformation approaches (e.g. VIATRA2 [BV06]) also allow to apply a rule on all matches in parallel in such a case.

Consistency recovery: Although EMF models show a graph-like structure and can be transformed similarly to graphs [EEPT06], there is a main difference between both concepts. In contrast to graphs, EMF models have a distinguished tree structure which is defined by the containment relation between their classes. An EMF model should be defined such that all its classes are transitively contained in the root class. Since an EMF model may have non-containment references in addition, the following question arises: What if a class which is transitively contained in the root class, has non-containment references to other classes not transitively contained in the root class? In this case, we consider the EMF model to be inconsistent, thus e.g. it cannot be persisted anymore.

A transformation can make an EMF model inconsistent, if its rule deletes one or more objects. For example an inconsistent situation occurs, if one of these objects transitively contains an object included by a non-containment reference. To restore the consistency, all objects to be deleted have to be determined. Thereafter, all non-containment references to these indicated objects have to be removed, too.

Similarly to the handling of deleted structures, consistency recovery is also applied to newly created objects. If a rule creates objects which are not contained in the tree structure, the consistency recovery will remove these objects at the end of a rule application. It is possible to forbid the application of those rules entirely, since inconsistencies on creation of objects can be determined statically.

5 Compiled Transformer Plug-ins

The third approach (illustrated in Fig. 3) for integrating model transformations over EMF models is the compilation of model transformation specification into a stand-alone transformer plug-in (e.g. a Java program), which takes a source EMF instance as input and generates a target EMF instance as output.

In the EMF context, such transformer plug-ins generated from graph transformation rules are basically Java programs, which manipulate EMF model instances via the domain-specific interfaces (APIs) generated by EMF. From a conceptual point of view, this compilation process needs to address the following main issues:

1. how to perform graph pattern matching efficiently in the case of single, parallel or as long as possible rule applications;

2. how to check the success or failure of pattern matching

3. how to manipulate models in a consistent and transactional way

It is well-known that the most critical step for the performance of graph transformation is the graph pattern matching phase. For this purpose, *constraint-solving techniques* and *search plan generation* are the two most frequently used and efficient strategies.

- Pattern matching can be formulated as a *constraint solving problem* where the LHS objects are variables, the objects of the EMF instance model form the domain and typing, linking und attribute values form the set of constraints.

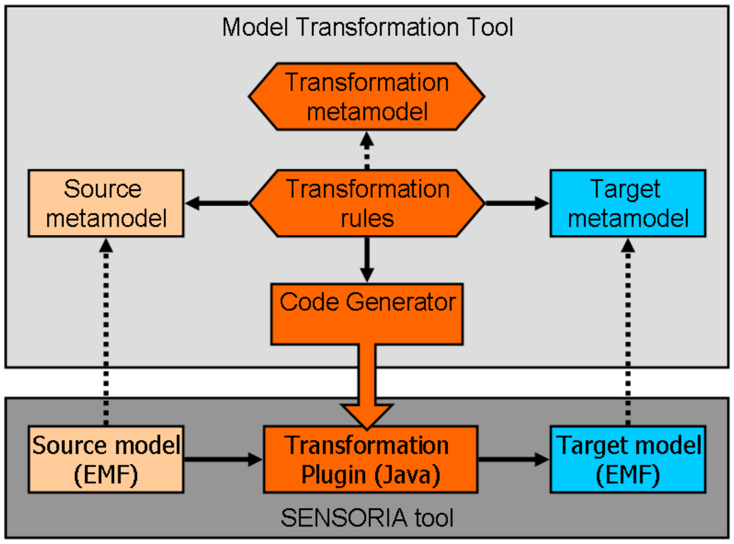

Figure 3: Transformer plugins in Java

- A *search plan* defines (at compilation time) the traversal order for the nodes of the instance model to check whether the pattern can be matched. Complex model-specific optimization steps can be carried out for generating efficient adaptive search plan as reported in [VVF05].

For rule application, Java code is generated which relies on those EMF classes already generated from the EMF model.

Transformation services by EJB3 transformer plugins. While EMF models are becoming increasingly popular, large service-oriented systems typically use other Java-based technologies providing persistent storage for business data and transaction handling for business transactions. Fortunately, the interfaces provided by EMF are very close to those Java-based techniques being e.g. Enterprise Java Beans (EJB), which are frequently used in service-oriented and web-applications. Therefore, we also investigated in [BVVP06, Var06] how leading industrial Java technologies (like EJB 3.0 [Sun06]) can be used as a platform for transformer plugins.

- **Compiling EJB3 models.** *EJB3 entity bean classes* will be generated from the source and target metamodels including the reference objects representing the mapping between them. The persistent storage of EJB3 entity beans will then be handled by the EJB3 application server.

- **Compiling EJB3 transformations.** *EJB3 session beans* can be generated from graph transformation rules in the form of fully functional EJB business methods. Transaction handling will be provided by the application server to prevent complex transformations from introducing conflicts when manipulating the model in parallel.

- **Compiling EJB3 search plans.** The concepts of search plans were adapted to the underlying database technology by translating graph pattern matching problems into database-independent EJB-QL queries, which reduce the memory consumption of traversing large model graphs.

Our measurements in [BVVP06] have demonstrated that transformer plugins with an underlying persistent store outperform pure Java solutions considering the size of the graph model by enabling to transform several million graph objects.

Since model transformations carried out by EJB3 transformer plugins can be easily published as web services, they fit very well to the service-oriented SENSORIA approach.

6 Related Work

There are several model transformation tools based on the Eclipse technology on the one hand, and on graph transformation techniques on the other hand. In [TEG$^+$05] we present a comparative study about different model transformation by graph transformation approaches. Furthermore, we started a first comparison with QVT [QVT05], the proposal for model transformation given by the OMG. In the following we distinguish between QVT-related and graph transformation related tools.

QVT-Related Tefkat [LS05] implements a declarative model transformation language for the transformation of MOF models using patterns and rules. It is implemented as an Eclipse plugin using the Eclipse Modeling Framework (EMF) to handle models based on MOF, UML2, and XML Schema. Unlike XSLT, Tefkat has a simple and familiar looking SQL-like syntax. It is specifically designed for writing scalable and re-usable transformation specifications using high-level domain concepts rather than operating directly on XML syntax.

ATL [ATL06] is a model transformation language specified both as a metamodel and as a textual concrete syntax. It is a hybrid of declarative and imperative approaches. The preferred style of transformation writing is declarative, which means simple mappings can be expressed simply. However, imperative constructs are provided so that some mappings which are getting too complex to be declaratively handled, can still be specified. Once complex mapping patterns are identified, declarative constructs can be added to ATL in order to simplify the writing of transformation systems. An IDE has been developed for ATL on top of Eclipse: ATL Development Tools (ADT), which uses EMF to handle models.

The Merlin [Mer06] Eclipse plug-in is based on the EMF JET Templates & Mapping model whose goal is to easen the process of automating the code generation and model transformation.

The Model Transformation Framework (MTF) [MTF05] is a set of tools that helps developers to make comparisons, check consistency, and implement transformations between Eclipse Modeling Framework (EMF) models. The framework also supports persistence of a record of what was mapped to what by the transformation; this record can be used to support round-tripping, reconciliation of changes, or display of the results to a user.

The MOMENT project contains a QVT-like EMF transformation engine which is based on algebraic specifications as implemented in Maude [BCR06]. This approach has a clear formal background which can be used for verification, but does not yet provide a visual definition of transformations.

In contrast to most of the related approaches, EMF transformations can be given a formal background by graph transformation which can be advantageously used to analyze model transformations. Analysis techniques include conflicts and dependencies analysis of rule applications, termination of model transformations and constraint checking.

Graph Transformation Related The Visual Modeling and Transformation System (VMTS) [LLMC04] is an n-layered metamodeling environment designed together with model transformation functionalities. VMTS is offering capabilities for specifying visual languages applying metamodeling techniques.

The AToM^3 [dLV02] tool allows the specification of Domain Specific Visual Languages by means of meta-modelling, and their manipulation by means of graph transformation. Recently, AToM^3 has been provided with the possibility to define triple graph grammars [GdL04] and multiple views [GDdL05]. Views are provided with their own meta-model, which optionally can be a subset of a global meta-model that relates all the view concepts. This is very useful when defining multi-view languages (specially if such views contain overlappings), such as UML.

GReAT (Graph Rewriting And Transformation) [GRe06] is a metamodel based graph transformation language useful for the specification and implementation of model-to-model transformations.

MoTMoT [Mot06] stands for Model driven, Template based, Model Transformer. It is a compiler from visual model transformations to repository manipulation code. The compiler takes models conforming to a UML profile for Story Driven Modeling (SDM) [FNTZ98] as input and outputs JMI code.

In [SG04] SDM has been used as a language for the visual development of refactorings (which are a particular kind of horizontal model transformations) and implemented in Fujaba [Fuj06] which suffers from two significant problems. First, the SDM metamodel in Fujaba is non-standard and it is only implicitly present in the source code. As a consequence, only the Fujaba editor is suitable to create and store SDM instances. Second, the Fujaba code generator exclusively generates code conforming to non-standard conventions, meaning it can solely be deployed on the Fujaba repository.

7 Conclusion

In this paper, we proposed three approaches for tool integration based on EMF model transformations. The following levels of tool integration were identified: (i) *model-level integration* carries out model transformations in existing transformation tools by importing and exporting EMF models, (ii) *interpreted EMF transformations* take a model of the transformation system, and manipulate EMF models according to the its rules by calling EMF interfaces, and finally (iii) *compiled transformer plugins* generate standalone transformer programs in Java which are responsible for model manipulation.

In SENSORIA, two main transformation tools, namely, TIGER [EEHT05] and VIATRA2 are planned to be used for tool integration purposes using our approach. While VIATRA2 is mainly a model transformation tool, the main purpose of TIGER is the generation of visual language environments from high-level specifications. Currently, the generation of visual editors and the execution of model transformations are supported. While VIATRA2 has support for approaches (i) and (iii), TIGER will have support for approaches (ii) and (iii), based on the EMF transformer developed in [BK06].

Acknowledgements

This work was partially supported through the IST-2005-16004 Integrated Project SENSORIA: Software Engineering for Service-Oriented Overlay Computers. The third author also received partial support from the János Bolyai Scholarship.

References

[AGG06] *AGG-System* `http://tfs.cs.tu-berlin.de/agg/`, 2006.

[ATL06] *ATL: The Atlas Transformation Language Home Page* `http://www.sciences.univ-nantes.fr/lina/atl`, 2006.

[BCR06] A. Boronat, J. Carsi, and I. Ramos. Algebraic Specification of a Model Transformation Engine. In *Springer LNCS 3922. Fundamental Approaches to Software Engineering (FASE'06). ETAPS'06. Vienna (Austria).*, 2006.

[BK06] Enrico Biermann and Günter Kuhns. Konzeption und Implementierung einer regelbasierten Transformationskomponente für das Eclipse Modeling Framework. Master's thesis, Technical University of Berlin, Department of Computer Science, 2006.

[BV06] András Balogh and Dániel Varró. Advanced model transformation language constructs in the VIATRA2 framework. In *ACM Symposium on Applied Computing — Model Transformation Track (SAC 2006)*, pages 1280–1287, Dijon, France, April 2006. ACM Press.

[BVVP06] András Balogh, Gergely Varró, Dániel Varró, and András Pataricza. Compiling model transformations to EJB3-specific transformer plugins. In *ACM Symposium on Applied Computing — Model Transformation Track (SAC 2006)*, pages 1288–1295, Dijon, France, April 2006. ACM Press.

[dLV02] J. de Lara and H. Vangheluwe. AToM3: A tool for multi-formalism and meta-modelling. In R.-D. Kutsche and H. Weber, editors, *5th International Conference, FASE 2002: Fundamental Approaches to Software Engineering, Grenoble, France, April 8-12, 2002, Proceedings*, volume 2306 of *Lecture Notes in Computer Science*, pages 174–188. Springer-Verlag, 2002.

[EEHT05] K. Ehrig, C. Ermel, S. Hänsgen, and G. Taentzer. Generation of visual editors as eclipse plugins. In *Proc. 20th IEEE/ACM International Conference on Automated Software Engineering*, IEEE Computer Society, Long Beach, California, USA, 2005.

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs in Theoretical Computer Science. Springer, 2006.

[EMF06] *Eclipse Modeling Framework (EMF)* `http://www.eclipse.org/emf`, 2006.

[EMO06] *Essential MOF (EMOF) as part of the OMG MOF 2.0 specification* `http://www.omg.org/docs/formal/06-01-01.pdf`, 2006.

[FNTZ98] T. Fischer, J. Niere, L. Torunski, and A. Zündorf. Story Diagrams: A New Graph Rewrite Language Based on the Unified Modeling Language and Java. In *In Proceedings of the 6th International Workshop on Theory and Application of Graph Transformation (TAGT), volume 1764 of LNCS, pages 296-309*. Springer, 1998.

[Fuj06] *Fujaba Project*, 2006. Available at `http://www.fujaba.de`.

[GDdL05] E. Guerra, P. Díaz, and J. de Lara. A Formal Approach to the Generation of Visual Language Environments Supporting Multiple Views. In *Proc. IEEE Symposium on Visual Languages and Human-Centric Computing, IEEE VL/HC C. Dallas*, 2005.

[GdL04] E. Guerra and J. de Lara. Event-Driven Grammars: Towards the Integration of Meta-Modelling and Graph Transformation. In *In Proc. ICGT'04 (Rome). LNCS 3256, pp.: 54-69. Springer*, 2004.

[GRe06] *GReAT: Graph Rewriting And Transformation `http://www.isis.vanderbilt.edu/Projects/mobies/downloads.asp`*, 2006.

[LLMC04] T. Levendovszky, L. Lengyel, G. Mezei, and H. Charaf. Systematic Approach to Metamodeling Environments and Model Transformation Systems in VMTS. In *2nd International Workshop on Graph Based Tools (GraBaTs), workshop at ICGT 2004, Rome, Italy*, 2004.

[LS05] M. Lawley and J. Steel. Practical Declarative Model Transformation With Tefkat. In *In Proc. Model Transformation in Practice Workshop, Models Conference*, 2005.

[Mer06] *Merlin Generator `http://sourceforge.net/projects/merlingenerator/`*, 2006.

[Mot06] *MoTMoT: Model driven, Template based, Model Transformer `http://www.fots.ua.ac.be/motmot/index.php`*, 2006.

[MTF05] *IBM Model Transformation Framework `http://www.alphaworks.ibm.com/tech/mtf`*, 2005.

[QVT05] *Query/View/Transformation (QVT). QVT-Merge Group, version 2.0 (2005-03-02). http://www.omg.org/cgi-bin/apps/doc?ad/05-03-02.pdf*, 2005.

[SG04] Hans Schippers and Pieter Van Gorp. Standardizing SDM for Model Transformations. In *Second Int. Fujaba Days (FD04), Darmstadt (Germany)*, 2004. Available at `http://www.lore.ua.ac.be/refactoringProject/publications/StandardizingS%DMforModelTransformations.pdf`.

[Sun06] Sun Microsystems. *Enterprise Java Beans 3.0*, 2006. `http://java.sun.com/products/ejb/docs.html`.

[TEG$^+$05] G. Taentzer, K. Ehrig, E. Guerra, J. de Lara, L. Lengyel, T. Levendovsky, U. Prange, D. Varro, and S. Varro-Gyapay. Model Transformation by Graph Transformation: A Comparative Study. In *Proc. Workshop Model Transformation in Practice*, Montego Bay, Jamaica, October 2005.

[Var06] Gergely Varró. Implementing an EJB3-specific graph transformation plugin by database independent queries. In *Proc. of the Fifth International Workshop on Graph Transformation and Visual Modelling Techniques*, ENTCS, pages 115–126. Elsevier, 2006.

[VIA06] *VIATRA2 (VIsual Automated model TRAnsformations) framework* `http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~/gmt-home/subpro%jects/VIATRA2/index.html`, 2006.

[VP04] Dániel Varró and András Pataricza. Generic and meta-transformations for model transformation engineering. In T. Baar, A. Strohmeier, A. Moreira, and S. Mellor, editors, *Proc. UML 2004: 7th International Conference on the Unified Modeling Language*, volume 3273 of *LNCS*, pages 290–304, Lisbon, Portugal, October 10–15 2004. Springer.

[VVF05] Gergely Varró, Dániel Varró, and Katalin Friedl. Adaptive graph pattern matching for model transformations using model-sensitive search plans. In G. Karsai and G. Taentzer, editors, *GraMoT'05, International Workshop on Graph and Model Transformations*, ENTCS Vol. 152, 2005.