Weaving Models with the Eclipse AMW plugin

Marcos Didonet Del Fabro, Jean Bézivin, Patrick Valduriez

ATLAS Group, INRIA & LINA
University of Nantes, Nantes, France
{marcos.didonet-del-fabro, jean.bezivin}@univ-nantes.fr,
patrick.valduriez@inria.fr

Abstract. The basic assumption in model engineering (MDE) is to consider models as first class entities. One of the most important kinds of models in MDE approaches are transformation models. Transformation models define operations between different models. However, there are many operations that are not efficiently handled by generic model transformations. For example, models transformations are not adapted to define and to capture relationships between models elements. Relationships between model elements are present in many different application scenarios, such as specification of transformations, traceability, or model alignment. We propose the use of weaving models to capture relationships between model elements. Weaving models can be used in different application scenarios, because they conform to different extensions of a core weaving metamodel. In this paper, we explain in detail what a model weaving is. We present a set of application scenarios, and we extract a set of basic requirements for creating relationships between model elements. Based on that, we define a core weaving metamodel, metamodel extensions, and describe a set of methods to create weaving models. We implement an Eclipse plugin named AMW (ATLAS Model Weaver) to support the creation of these weaving metamodels and models.

1 Introduction

Within the Eclipse Modeling Project GMT research incubator there is a plugin named AMW (ATLAS Model Weaver) with an associated user community. The present paper presents the rationale for using this plugin, with a set of application scenarios that motivate the plugin creation. AMW provides an adaptive workbench to support different applications of weaving models.

The basic assumption in model driven engineering (MDE) is to consider models as first class entities. A model is an artifact that conforms to a metamodel and that represents a given aspect of a system. Current MDE approaches usually have three representation levels for models: metametamodel, metamodel and terminal models [Jouault 2006]. The metamodel describes the various kinds of the elements of a model and the way they are arranged, related and constrained. The metametamodel is the base representation format of all metamodels and models of one technical space [Bézivin 2005].

MDE platforms are composed of different kinds of models. One of the most important kinds of models are transformation models. Transformation models are used to define operations between model elements. A transformation model defines how a set of input models is transformed in a set of output models. A transformation model is executed in a transformation engine.

However, model transformations are essentially conceived to define executable model transformation operations. An important issue that has not been studied in details by MDE solutions is how to define relationships (or links) between model elements, and to use them in practice. Links between models are used in many different classes of problems. In ontology merging [Maedche 2002, Mitra 2000], ontology bridges specify the equivalences between concepts of two or more ontologies. In data translation [Miller 2001, Milo 2000], one-to-one correspondences establish links between schema elements. These correspondences are used to automatically produce transformations. In model merging [Pottinger 2003], a set of links are used as input for a generic merging algorithm. In tool interoperability, weaving models are used as specification for tool interoperability transformations. In traceability scenarios [Jouault 2005], a traceability model keeps track of the source models elements that are used to produce a set of target elements. In model composition [Bézivin 2006], a set of links can specify how two models are composed.

We propose to use weaving models to capture different kinds of links between model elements. A weaving model conforms to a weaving metamodel. Weaving models have special characteristics. They are not self contained, i.e., a weaving model is useful only if the related models exist as well. The links have different semantics, depending on the application scenario. For instance, an ontology merge link has a different semantic than a traceability link. Thus, the weaving metamodel must be extensible to capture these different kinds of links.

Weaving models are created using different methods. The method used is closely related to the corresponding application scenario. For instance, matching heuristics can be used to automatically create weaving models that are specification for model transformations. Weaving models can be created manually by graphical user interfaces. Transformations can automatically generate weaving models to produce traceability information.

Despite having large number of possible semantics, there is a set of common features in almost all of these application scenarios. We specify a core weaving metamodel that factors out these features. This metamodel provides basic link management. The different kinds of links are created as separated domain specific weaving metamodels (DSWMs), which are extensions to the core weaving metamodel. This is of significant importance, because as for normal models, the definition of a complete and complex metamodel capable of handling every application scenario is not a practical solution.

In this paper, we present in detail what a weaving model is. First, we present a set of application scenarios in which is necessary to establish links between model elements. Based on these scenarios, we describe a set of common requirements for link management. Then, we describe the core weaving metamodel, and the key features to handle with these requirements. Finally we describe different methods used to create weaving models with the help of the generic AMW Eclipse plugin.

This paper is organized as follows. Section 2 describes the set of application scenarios. Section 3 presents the common requirements for link management. Section 4 describes a correspondence metamodel to capture these features. Section 5 describes different methods to create weaving models. Section 6 concludes.

2 Application Scenarios

In this section we describe a set of application scenarios where it is necessary to establish different kinds of links between model elements. First, we describe how links can be used to capture the semantic heterogeneities in tool interoperability scenarios. Second, we show how model composition operations are described in terms of typed-links. Then we present a traceability scenario. Finally, we introduce the use of different kinds of links in model alignment scenarios.

2.1 Tool Interoperability

Tool interoperability aims at using the models produced by one tool into another tool. Consider that two or more tools have distinct data, but they cannot come into agreement into a common tool or metamodel, due to practical reasons. In this case, it is necessary to produce transformations of one tool model (a source tool) into another tool model (a target tool).

It is necessary to capture all the semantic heterogeneities between the set of tools and to translate the models from the source tool into the target tool. Many different transformation patterns are applied in these transformations, such as equality, equivalence, concatenation, etc.

These transformation patterns depict translation links between the source and target tool metamodels. Each one of these patterns is usually coded in some specific transformation language. However, instead of directly creating a transformation between the tool models, typed links can capture these semantic heterogeneities in a more abstract representation.

This has some advantages. The links can be created independently of any model transformation language. This enables to abstract any implementation details, such as complex navigation expressions. These links are semantically closer to the application domain than generic model transformation languages. They can be created by domain experts using adapted graphical user interfaces, or by automatic executing matching heuristics. Matching heuristics exploits the data from the source and target tool metamodels to automatically capture all (or a set) of the links between the tool models.

This enables to semi-automate the process of transformation development for tool interoperability. These links are finally translated into an executable transformation language.

2.2 Model Composition Operation

Model composition is an operation that combines two or more models into a single one. A composition operation is a special type of transformation that takes two models *Ma* and *Mb* as input and that combines their elements into a model *Mab*. Model composition is a generic operation that varies from application to application. Which elements from *Ma* and *Mb* are combined and in which way depends on the operation implementation. For instance, if we compose two relational schemas, not all tables from both schemas are necessarily in the resulting schema. Composition operations contain different kinds of link that can be established, such as inheritance, aggregation, merging, overriding, union, etc.

In the same way as in tool interoperability scenarios, there are different kinds of composition semantics. One of the major differences is relative to the cardinality of links, because typical composition scenarios have two models as input and one model as output. The abstraction level between the link specification and the final generated transformation is higher than in tool interoperability scenarios. General purpose transformation languages usually do not support the different kinds of composition links in a straightforward way.

The links are high-level specifications for composition operations. The links are transformed into specific composition code patterns. How these patterns are implemented is not relevant to the operation developer. Consider for example one composition link called *Union.* The operation developer that creates a *Union* link is not aware about the transformation that is used to execute this low level operation.

In addition, not all kind of models support every composition links. For instance, a relational model does not have inheritance or aggregation implemented natively. This motivates the creation of separated sets of links used only in dedicated composition applications. After these links have been established, we should generate transformations to execute the composition.

2.3 Traceability

Traceability aims at maintaining the connection between a set of models that are part of a complex process. For instance, traceability is related to model evolution or model provenance. Many different operations may be executed over models: they are transformed, updated, re-factored, composed, etc. Traceability aims at storing this information. For example, when transforming one model into another, traceability information enables to identify the set of target elements that are used to generate a set of target elements.

Thus, traceability is achieved by creating different kinds of trace links between models (or metamodels) elements. The kinds of links that are created vary, for example *Added*, *Modified*, *Generated*, *Replaced*, etc. There are different use cases for traceability, for instance requirements traceability. Requirements traceability keeps track of all the steps of a development process, analysis, design, programming, testing, etc. The kinds of links are *developed_by*, *allocated_to*, *performed*, *based_on*, *modify,* etc [Ramesh 2001].

2.4 Model Alignment

Model alignment is the establishment of links between models that are semantically close, i.e., that represent a similar domain. Differently from model composition scenarios, model alignment does not merge or compose the linked models.

Common use cases for model alignment are ontology alignment scenarios. In a second extend, the links can be used as input for generic merging operations, or to produce simple translation languages similar to tool interoperability scenarios.

The kinds of links usually indicate the semantic proximity between the linked elements. For instance, links may indicate equivalence relationships, equality, similarity, negation, exclusion.

3 Common Requirements for Link Management

Despite having different objectives and applications, the basic notion of every application scenario described in the previous Section is the notion of a link. We illustrate the common requirements to establish links between model elements using two models *Ma* and *Mb*. Consider a weaving model *Mw* between *Ma* and *Mb*, denoted by the triple <*Mw, Ma, Mb*>. *Mw* contains a set of elements that link a set of elements of *Ma* with a set of elements of *Mb*.

We propose to capture different kinds of links between model elements in weaving models. Weaving models have different purposes, depending on the kind of links required. For example a merge weaving model indicates how to merge elements of two models. A traceability weaving model indicates that one model is a new version of another.

Let us assume now that *Mw* is a merge weaving model. An element $mw \in Mw$ indicates that a given element $ma \in Ma$ and an element $mb \in Mb$ are linked. These elements are going to be merged as a result of a merge operation. In this case merge link is motivated by the equivalence semantics between the linked elements. The same is valid if *Mw* is a traceability weaving model. However, the link is interpreted in a different way: *mb* is derived from *ma* after some update operation.

Hence, the links have a meaning, denoted by their type. For instance, in the merge scenario the link between elements *ma* and *mb* may conform to a link type denoting equivalence. In the traceability scenario, a *ma* may be new version of *mb*.

A link may relate elements from more than two models. Consider a third model *Mc*, a tuple <*Mw, Ma, Mb, Mc*> and a model element $mc \in Mc$. An element $mw \in Mw$ has a different semantic. For example it may indicate that *ma*, *mb* and *mc* are added and their result divided by three *((ma + mb + mc)/3)*. Finally, the weaving model *Mw* does not contain the concrete elements *ma, mb* or *mc*, but a reference that enables to access them in the containing models (*Ma*, *Mb* and *Mc*). It is necessary to have a way to access and to uniquely identify each linked element.

Based on that, we define four main requirements to support link management using weaving models:

- A weaving model should express the notion of links between model elements.

- Different links types must be supported. The link type provides the semantic information on how the elements are related. For instance a link may indicate the equality or concatenation of model elements.
- It must be possible to define links with different arities (unary, binary, ternary, etc.), i.e., a link has many endpoints. For example an equality link has two endpoints, while a concatenation link may have three endpoints (to elements concatenated into one).
- The weaving model should have an identification mechanism to uniquely identify the model elements. The link endpoints do not contain the concrete metamodel elements, but a pointer that enables to access them in the containing models.

4 Weaving Metamodel

We define a core weaving metamodel to support the common requirements for link management. The design of a core weaving metamodel is a delicate compromise between expression power and minimality.

We start by defining the core weaving metamodel. Then we present a set of possible extensions to this core metamodel.

4.1 Core Weaving Metamodel

We illustrate the core weaving metamodel in Figure 1. This metamodel is introduced in [Didonet Del Fabro 2005].

- *WElement* is the base element from which all other elements inherit. It has a name and a description.
- *WModel* represents the root element that contains all model elements. It is composed by the weaving elements and the references to woven models.
- *WLink* fulfils the first and second requirements. *WLink* express a link between model elements, i.e., it has a simple linking semantics. To be able to express different link types and semantics, this element is extended with different metamodels (we explain how to add different link types in the following section).
- *WLinkEnd* handles the third requirement (called link endpoint). Every link endpoint represents a linked model element. This way it is possible to create N-ary links.
- *WElementRef* satisfies the fourth requirement. We associate the *WElementRef* element with an identification function over the related elements. The function takes as parameter the model element to be linked and returns a unique identifier for this element. For practical reasons we define a string field ref that saves the return value of the function. There is also the inverse function that takes the value of the ref attribute, and returns the element from the related model.
- *WModelRef* is similar to *WElementRef* element, but it references an entire model.

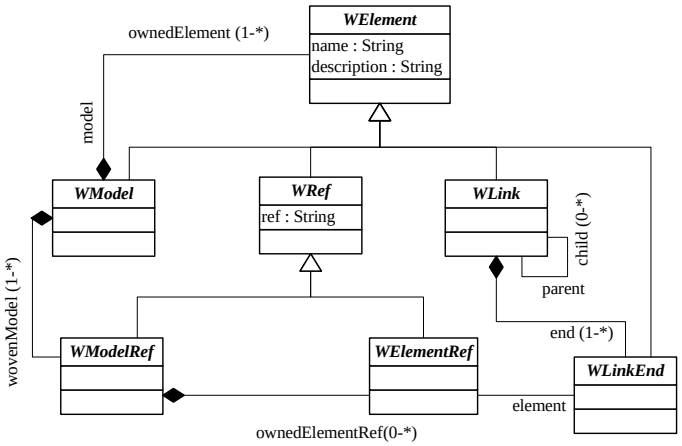

Fig. 1. The core weaving metamodel

One may say it is possible to associate the identification functions directly with the link endpoints. We create separated *WElementRef* because it enables referencing the same model element by several link endpoints.

This metamodel has only abstract types. However, we illustrate a simple weaving model in Figure 2. It links the elements of *LeftMM* and *RightMM* metamodels. The weaving model contains one link (*WLink*); the link contains two endpoints (*WLinkEnd*), i.e., one refers to an element in *LeftMM* and the other to an element in *RightMM*. Each *WLinkEnd* refer to one *WElementRef*.

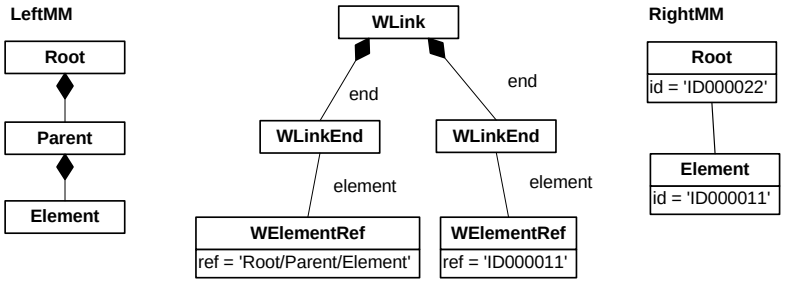

Fig. 2. A simple weaving model

The left *WElementRef* has the identification (ID) of Element from *LeftMM*. The ID is calculated taking the element name (*Element*) and the name of the parents (Root/Parent). The right *WElementRef* refers to Element from *RightMM*. The ID is a string that is automatically generated. In this example the number of endpoints and linked elements are the same. However, we do not set the element ID directly in the *WLinkEnd* element because this way it is possible to refer the same element by

different endpoints. Different link types, link endpoints and identification mechanisms are added using metamodel extensions. The link element must be extended to create different link types, for example equality, equivalence, dependency, and so on.

4.2 Example of Concrete Weaving Types

As already stated, the core weaving metamodel is not designed to handle every kind of link. In this section we describe different subsets of domain specific weaving metamodels DSWMs that are extensions of the core weaving metamodel.

The definition of different link types is not a trivial task. It is an application oriented task that often requires in depth knowledge of the underlying domain. We envisage different DSWMs with different types of links:

- Composition: links such as *Override*, *Merge*, *Delete*.
- Interoperability: links such as *Equality*, *SourceToTarget*.
- Data integration: *Concatenation*, *Equality*, *IntToStr*.
- Traceability: *Origin*, *Source*, *Evolution*, *Modified*, *Added*.
- Ontology alignment: *Equivalent*, *Equality*, *Resemblance*, *Proximity*.

From this list (which is not exhaustive) we can see that some type of links overlap between different domains, for example equality links are available in almost every scenario. This motivates the creation of different sets of modular extensions to the core weaving metamodel. The extensions are reused in different application to finally create the desired DSWMs. We illustrate a set of extensions in Figure 3.

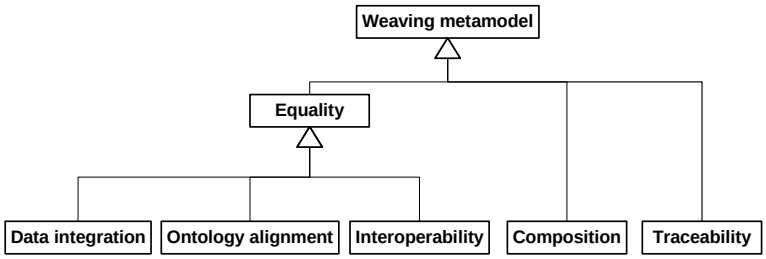

Fig. 3. A set of DSWMs extensions

A complete weaving metamodel covering every case is not a practical solution. The set of possible links is very extensive, adding much complexity and many link types that are not going to be used only in specific scenarios.

5 Creation of Weaving Models

In this section we introduce different methods to create weaving models. The whole process of creation of weaving models is encapsulated in a model management operation called *Match* [Bernstein 2003]. The *Match* operation is semi-automatic, i.e., it is an interactive process that alternates between the automatic execution of matching heuristics and the manual refinement of weaving models in a weaving tool.

5.1 Automatic Generation

The automatic generation of weaving models is subsumed in a match operation. The *Match* operation takes two models M_a and M_b as input and produces a weaving model M_w as output. M_a and M_b conform to MM_a and MM_b; M_w conforms to MM_w.

$M_w : MM_w = Match\ (M_a : MM_a, M_b : MM_b)$.

The match operation creates a weaving model conforming to an extended weaving metamodel. The match operation executes a set of matching heuristics to search for similar concepts in the input models. There are many different methods that can be used to automatically create weaving models. We explain below two different methods: element-to-element and structural methods.

5.1.1 Element-to-element

Element-to-element methods calculate a similarity value between the elements of the input models. These values are used to create links between these elements. A link with a high similarity value indicates that there is a good probability that these elements have some semantic resemblance. Element-to-element methods consider only information about a pair of elements. We explain two different methods:

- *String similarity*: the names of the model elements are considered strings. The names are compared using string comparison methods such as Levenshtein distance and edit distance [Cohen 2003].
- *Dictionary of synonyms*: the names are compared using a dictionary of synonyms (we use WordNet [Fellbaum 1998]). This dictionary provides a tree of synonyms. The similarity between two terms (element names) is calculated according to the distance between these terms in the synonym tree.

5.1.2 Structural

Structural methods calculate similarities using the internal properties of the model elements, e.g., types, cardinality, and the relationships between model elements, e.g., containment or inheritance trees. These data are encoded in the metamodels. They improve the accuracy of element-to-element methods.

- *Internal properties*: model elements have a set of properties, such as type, cardinality, order, length, etc. Consider two model elements $a \in Ma$ and $b \in Mb$; *Ma* and *Mb* are different models, but conform to the same metamodel. The set of properties of *a* are compared with the set of properties of *b*. If a given property has the same value, the elements have a good probability to be similar.
- *Element relationships*: there are different kinds of relationships between elements of the same metamodel, for instance containment or inheritance relationships. Structural methods that exploit the element relationships rely on the following assumption: if two model elements are similar, the neighbors of these elements are likely to be similar as well. For example, if two attributes from two models have a high similarity value, the containing classes of these attributes have a good probability to be similar. One example of algorithm that exploits structural information is the similarity flooding algorithm [Melnik 2004].

5.2 Manual Creation

Weaving models can also be created manually. For this it is necessary to have a tool that adapts to different kinds of metamodel extensions. We describe our tool called the ATLAS Model Weaver. The tool is available for download as an Eclipse GMT subproject [AMW 2006]. The tool provides mechanisms for manual creation of weaving metamodels and weaving models. The tool reuses part of the infrastructure of the ATL IDE [ATL 2006] based on the Eclipse Platform. The three notions on which we based the design are metamodel extensions, Eclipse Plugins and the Eclipse EMF platform [EMF 2006] for models' manipulation. The tool borrows engineering concepts from the Eclipse Platform: to build a base workbench that is extensible to a wide range of applications.

The workbench defines itself different extension points to contribute to the main editor (see Figure 4). The workbench is responsible for controlling the interaction between the different plugged components. The main idea of the implementation is to have a simple user interface of the weaving plugin that may be partially generated, without having to build a specific tool for each weaving metamodel.

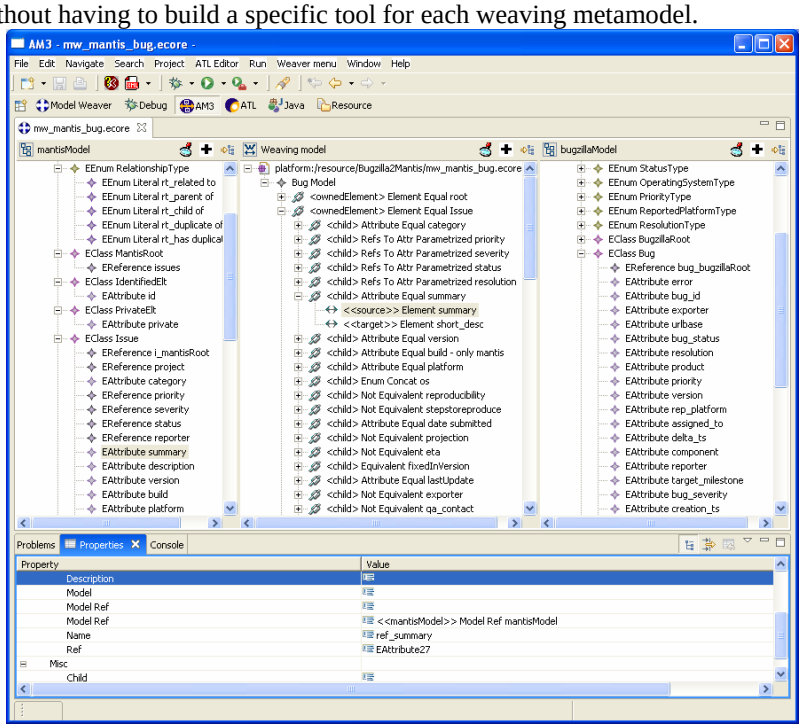

Fig. 4. The AMW plugin

The plugin handles the base weaving metamodel described in the Section 4 and it can be extended incrementally. It is initially composed of three components: a left metamodel, the weaving metamodel and a right metamodel. The standard functionalities are themselves added as an extension to the base plugin, thus validating our generic approach.

The screen-shot of Figure 4 is formed by 3 panels. The panels from left and right are tree interfaces extending the standard interface provided by EMF. The weaving model conforms to a weaving metamodel. The left and right panels can be exchanged, for instance by three-like panels or by graphical interfaces with boxes to represent elements and lines to represent links. The only constraint is that the components may implement a previously defined interface to return all model elements such that they can be accessed by the weaving component.

6 Conclusions

In this paper, we have presented a detailed description of model weaving. We described what a weaving model is, and how it conforms to extensions of a core weaving metamodel. This core weaving metamodel supports basic link management. We have shown from a set of application scenarios that the creation of links between model elements is an important issue in MDE.

The diversity and specificity of the application scenarios demonstrate that it is not possible to efficiently cover every application requirement by using generic mechanisms, such as generic transformation languages, or mapping models. We presented a model weaving Eclipse plugin (ATLAS Model Weaver) that uses the reflective API of EMF to adapt to different weaving metamodel extensions.

The metamodel extensions enable to create weaving metamodels with a vocabulary closer to each application domain. This allows creating weaving models using adapted graphical user interfaces, or using semi-automatic methods, such as different matching heuristics. These semi-automatic methods bring many advantages to the general process of transformation development and model alignment, since the semantic relationships are fully (or partially) discovered by automatic methods.

There are some challenges yet to solve, for instance how to ameliorate existing matching methods to become more and more accurate, thus diminishing the human intervention on the link creation. Another important issue is to create different subsets of weaving metamodel extensions that subsume the most frequently used patterns for each application scenario, leading to the standardization of the domain.

There is current a user community for the Eclipse AMW plugin. As new applications are found, it is very likely that this community will grow in the future.

Acknowledgments

This work is partially supported by the Modelplex project. We would like to thank the members of the ATLAS team for all the help, and especially Frédéric Jouault and Ivan Kurtev.

References

[AMW 2006] AMW: The ATLAS Model Weaver. Ref. site: http://www.eclipse.org/gmt/amw, 09/2006

[ATL 2006] ATL: ATLAS Transformation Language. Ref. site: http://www.eclipse.org/gmt/atl, 09/2006

[Bernstein 2003] Bernstein, P A. Applying Model Management to Classical Meta Data Problems. In proc. of CIDR 2003, pp 209-220

[Bézivin 2005] Bezivin, J, Kurtev, I. Model-based Technology Integration with the Technical Space Concept, In Metainformatics Symposium 2005, Esbjerg, Denmark, November 2005, to be published in LNCS volume

[Bézivin 2006] Bézivin, J, Bouzitouna, S, Didonet Del Fabro, M, Gervais, M P, Jouault, F, Kolovos, D, Kurtev, I, Paige, R F. A Canonical Scheme for Model Composition, In Proc. European Conference in Model Driven Architecture (EC-MDA) 2006, Bilbao, Spain, July 2006

[Cohen 2003] Cohen, W, Ravikumar, P, Fienberg, S E. A Comparison of String Distance Metrics for Name-Matching Tasks. In proc. of IIWeb 2003, pp 73-78

[Didonet Del Fabro 2005] Didonet Del Fabro, M, Bézivin, J, Jouault, F, Valduriez, P. Applying Generic Model Management to Data Mapping. In proc. of Base de Données Avancées (BDA 2005), 17-20/10/2005, Saint-Malo, France

[EMF 2006] EMF. Eclipse Modelling Framework. Reference site: http://www.eclipse.org/emf, 09/2006

[Fellbaum 1998] Fellbaum, C. WordNet, an Electronic Lexical Database. MIT Press, 1998. Reference site: http://wordnet.princeton.edu/

[Jouault 2005] Jouault, F. Loosely Coupled Traceability for ATL. In: Proceedings of the European Conference on Model Driven Architecture (ECMDA) workshop on traceability, Nuremberg, Germany. 2005

[Jouault 2006] Jouault, F, Bézivin, J. KM3: a DSL for Metamodel Specification. In proc. of 8th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems, LNCS 4037, Bologna, Italy, pages 171-185. 2006.

[Maedche 2002] Maedche, A, Motik, B, Silva, N, Volz, R. Mafra - a mapping framework for distributed ontologies. In proc. of EKAW 2002, pp 235-250

[Melnik 2004] Melnik, S. Generic Model Management: Concepts and Algorithms, Ph.D. Dissertation, University of Leipzig, Springer LNCS 2967, 2004

[Miller 2001] Miller, R J, Hernandez, M A, Haas, L M, Yan, L-L, Ho, C T H, Fagin, R, Popa, L. The Clio Project: Managing Heterogeneity. In SIGMOD Record 30, 1, 2001, pp 78–83

[Milo 1998] Milo, T, Zohar, S. Using Schema Matching to Simplify Heterogeneous Data Translation. In proc. of VLDB 1998, pp 122-133

[Mitra 2000] Mitra, P, Wiederhold G, Kersten M. A graph-oriented model for articulation of ontology interdependencies. LNCS, 1777:86+, 2000

[Pottinger 2003] Pottinger, R A, Bernstein, P A. Merging Models Based on Given Correspondences. In proc. of VLDB 2003. Berlin, Germany, pp 862-873

[Ramesh 2001] Ramesh, B, Jarke, M. Toward Reference Models for Requirements Traceability. IEEE Transactions on Software Engineering. V27 , Issue 1 (January 2001), pp 58-93