Towards a unified view of model mapping and transformation

Paolo Bottoni[1] Fulvio D'Antonio[2] Michele Missikoff[2]

[1] Università "La Sapienza" di Roma - Dip. di Informatica
Via Salaria, 113 - 00198 Roma - Italy
`bottoni@di.uniroma1.it`
[2] LEKS - Lab for Enterprise Knowledge and Systems
IASI - CNR, Viale Manzoni 30, 00185 Rome (Italy)
{`missikoff|dantonio`}`@iasi.cnr.it`

Abstract. In the last period, there's an increasing demand for "model transformation" techniques (e.g. to support the Model Driven Development proposed by OMG). Tools and techniques for *model mapping*, *model transformation* and *model merging* are developed in several domains (from software engineering to ontology building); however the various initiatives often diverge in terminology or in the overall view of the mapping/transformation process, hindering the clear understanding or the reuse of the techniques themselves. With this paper we aim to propose a number of formal definitions of relevant concepts in the model mapping/transformation domain; we argue that this effort will be useful both for the user of transformation/mapping tools (for a better understanding of the task to be achieved) and for the tool's developer (allowing better and unambiguous description of the tool's functionalities).

1 Introduction

In the last period, there's an increasing demand for "model transformation" techniques. The main applications of model and meta-model transformations deal with language or formalism translation (e.g. translating an UML model to an OWL ontology), or information restructuring (e.g. renaming all occurences of the UML class "Person" with "Human being"). Another foreseen application is the support to automatic software generation, that in the last period has come to new life. The most relevant initiative in such direction is the Model Driven Development (MDD) proposed by OMG: MDD introduces an approach to system specification that separates the views on three different layers of abstraction: high level specifications of what the system is expected to do (Computation Independent Models or CIMs), the specification of system functionality (Platform Independent Models or PIMs) from the specification of the implementation of that functionality on a specific technology platform (Platform Specific Models or PSMs). The advantages advocated by MMD approach are the following :

– PIMs provide formal specifications of the structure and functions of the system that abstract from technical details,

– Implementation on several platforms can be done from the same PIM,
– System integration and interoperability can be anticipated and planned in the PIM but postponed to the PSMs regarding details
– It is easier to validate (the correctness of) models.

The core of the process is surely the transformation phase. Starting from a CIM a number of transformations must be performed to get a PIM and then down to a PSM .
OMG launched a request for proposal to work on the standard for the mapping-transformation issues; the language, to come, will be called QVT (Query-View and Transformation) .

On the other hand, a lot of work has been done in the field of "transformations-related" techniques. Lambda calculi, Post system, Term and Graph rewriting systems are examples of well-known and formal grounded computational systems that are referred to be: "transformation systems". Applications of the cited systems are for example

– theorem proving
– algebraic specification (of data types, programs etc.)
– λ-calculi
– implementation of declarative languages
– operational semantics of programming languages

In our perception, this rich heritage should be exploited to benefit both of a meaningful terminology and of a formal grounding for future transformation methods and tools proposals.

In this paper we concentrate, however, on the former objective, i.e. to establish a meaningful terminology, defining concepts belonging to the model transformation/mapping domain. We give, therefore, a definition of some relevant terms (such as *model transformation*, *model correspondence/mapping*, *model merging*, *correspondence/mapping discovery*) by giving also a mathematical characterization of them (essentially drawn from graphs and algebraic world).

The rest of the paper is organized as follow:

Section 2 presents the related work; in Section 3 preliminary mathematical notions are presented, while in Section 4 we discuss the graph formalism that could be used to represent different kind of models. Sections 5 and 6 give formal definitions about correspondences, mapping and transformation. Section 7 gives a characterization of *model correspondance/mapping discovery* and in Section 8 conclusions and future work are discussed.

2 Related Work

The idea of proposing an abstract view of model transformation has its roots in the algebraic approach to Graph transformation proposed by Ehrig *et al.* in [8], exploiting a definition of graphs as Σ-algebras, and the pushout construction in the category of graphs and total graph morphisms. Several developments have been originated from this pioneering work, both in the direction of including several classes of graphs into the algebraic approach (e.g. attributed [12, 13], distributed [7], hierarchical [17]), and in the direction of extending it to a more general categorical setting, where the relevant structure of a model is given by its categorical properties, rather than its specific organisation as a graph or otherwise [6, 5].

Moreover, algebraic approaches have been put to work on model transformation for several specific cases. Heckel *et al.* considered modeling of system oriented architectures and architecture transformations or matching, for example in the MDA perspective or for service discovery [10]. Transformations were also applied in a software engineering perspective for defining semantics of parts of the UML suite [21], for automatic code generation and verification [19] and for refactoring of models [18] and code [1].

Approaches based on direct coding of these transformations have also been exploited, (e.g. [16]), but the advantage of an algebraic approach is that it easily lends itself to demonstrations as far as properties of the transformations are concerned, related both to the transformation system *per se*, e.g. independence of transformations [14], termination [2, 4], and to the semantics of the specific application, once it is properly formalised [11].

3 Preliminary Notions

We recall here some basic algebra notions from [20].

Given two sets A and B, a *partial mapping* f is a subset of $A \times B$ (the usual notation is $f(x) = y$ instead of $(x, y) \in f$) such that $\forall x \in A$, $f(x) = y$ and $f(x) = z \Rightarrow y = z$. If $\forall x \in A\ \exists y \in B$ with $(x, y) \in f$, then f is called a (total) mapping and we write $f : A \rightarrow B$.

Given a set A and a natural number n, an *operation* is a mapping from A^n to A. An *algebra* (A, Φ) is a set A together with a family Φ of finitary operations. In general, one distinguishes the signature Σ, defining the name and arity (i.e. a value n) for each operation in Φ, from the realisation of the operations. Given a signature Σ, a Σ-algebra A is a pair $(\mathbf{A}, \Sigma^A)$, where $\mathbf{A}$ is the *carrier set* of A and Σ^A is the family of realisations for the operations in Σ.

Given two Σ-algebras A and B, a mapping $f : A \rightarrow B$ is called a *homomorphism* if $\forall \sigma \in \Sigma\ f(\sigma^A(x_1, \ldots, x_n)) = \sigma^B(f(x_1), \ldots, f(x_n))$. If carriers $\mathbf{A}$ and $\mathbf{B}$ are partially ordered and f preserves the ordering, then f is a *morphism*.

Given a signature Σ, and X a set of variables, a *term* is a construct of the form $x \in X$, or $\sigma(t_1, \ldots, t_n)$ for σ an operation of arity n and t_i a term for $i = 1, \ldots, n$. The set of terms constructed in this way is denoted $T_\Sigma(X)$. Given a signature Σ, an *equation* is a construct $\forall X(t = t')$, with $X \subset V$ (V a countably infinite set of variables), $t, t' \in T_\Sigma(X)$. An equation is *valid* in a Σ-algebra A if for all assignments α of symbols in X to values of $\mathbf{A}$, $\alpha^{\#}(t) = \alpha^{\#}(t')$, where $\alpha^{\#}$ indicates the canonical extension of α from variables to terms. An algebra A is a *model* for a set of equations E if A validates each equation of E.

In many cases, we are interested in carrier sets including values of different type. In this case, we speak of many-sorted algebras, by introducing the notion of *sort*. Then, each value of $\mathbf{A}$ belongs to a specific sort s from a set S. The arity of an operation in Σ is defined by a function $ar : \Sigma \rightarrow W(S) \times S$, where $W(S)$ is the set of all words of finite length built with symbols from S. The realisation of an operation must then be consistent with its declared arity. All the above notions for Σ-algebras are immediately transferable to many-sorted Σ-algebra.

Graphs are finite collection of nodes and edges and are often used as models of some domain by considering nodes as representations of some domain entity and edges as relations between them. In order to have a formal account of this usage, we define a *graph* as a multisorted Σ-algebra with $S = \{node, edge\}$, where Σ includes a finite set of nullary operations (constants), each creating a node and operations *source* and *target*, both of arity $(edge, node)$.

In general, we are interested in graphs with more complicated structure, as can be provided by the use of labels or of attributes, and consequently of types. In particular, a labelled graph is defined by including two sorts $label_E$ and $label_N$ in S, two sets L_N and L_E in the carrier $\mathbf{A}$, and operations $lab_N : node \rightarrow label_N$ and $lab_E : node \rightarrow label_E$ in Σ. For attributes, one employs a graph signature, as above, and a data signature.

4 Model Representation

In this section and in the rest of the paper the term "model" is used according to the following definition: "A model is a set of statements about some system under study"[15].

However, this is not in contrast with the definition in Section 3, as these statements can be seen as ground equations in the definition of the algebra, and the algebra operations can be seen as ways to infer new statements.

Several kind of models (e.g. Enterprise models, Software models, Process models) and modelling languages exist and a common way to present them to a human user is by means of a diagrammatic representation. This is possible due to the intrinsic “graphic” (see Figure 1) nature of most of modelling languages; in fact, most of them can be represented using nodes and edges connecting them; moreover nodes and edges can have a label associated with them. We have choosen therefore to adopt a powerful graph formalism as a base for representing different kind of models and modelling languages.

Definition 1. *A* **Labelled Directed Multigraph (LDMGraph)** *is a 6-tuple $G = (V, E, s, t, l_v, l_e)$, where V, E are two finite sets, $s, t : E \rightarrow V$ are two functions indicating source and target of an edge, and $l_V : V \rightarrow \Sigma_V$, $l_E : E \rightarrow \Sigma_E$ are two functions from V and E in the two finite sets of labels Σ_V e Σ_E.*

Definition 2. *Given a model M, $EL_M = V \cup E$ is the set of the basic elements of the model (that is, nodes and edges). Given a graph $\mathcal{G}$, we define the set of all subgraphs of $\mathcal{G}$ as $Sub(\mathcal{G})$.*

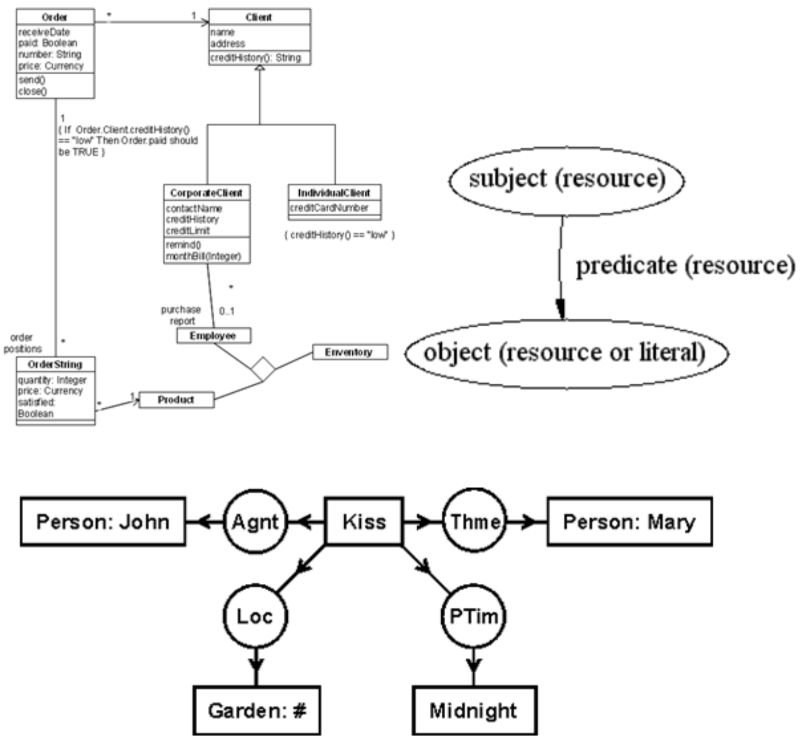

Fig. 1. "Graphic" nature of popular modelling languages: UML, RDF/OWL, Conceptual Graphs

In the rest of the paper a subscript will be used to denote the elements of a tuple defining an LDMgraph G (e.g. V_G is the set of vertexes of the LDMgraph G etc.)

Definition 3. *An* **LDMgraph** *G is said to be typed if exists an LDMgraph T such that l_{TV} and l_{TE} are bijections, and a function type: $EL_G \rightarrow EL_T$ (mapping nodes in nodes and edges in edges) and $\forall e \in E_G$ $type(s_G(e)) = s_T(type(e))$ and $type(t_G(e)) = t_T(type(e))$.*

We call **MOD** the set of LDMgraphs and **M2M** is the set of functions f such that f: **MOD**$\rightarrow$**MOD**; given a family of models M and a function mc: $M \rightarrow MOD$ then $M_{mc} = \{mc(x) : x \in M\} \subseteq MOD$ is the subset of LDMgraphs that are representation of the models in M; e.g. given the set of **UMLModels** and a procedure **umlmc** to convert the concrete syntax of UML serialization (for instance XMI) to an LDMgraph representation, then $UMLMODELS_{umlmc}$ is the set of graphs in MOD that are representation of some UML model.

We assume that the translation from concrete syntaxes of existing modelling languages into the proposed graph formalism is a not difficult task; we give an example by representing the metamodel of UML state diagram in Figure 2. In such picture every state(transition) is represented by a node of an attributed graph, labelled with the name of the state(transition) and every transition is linked to states by edges labelled with "source" and "target".

5 (Inter-)Model correspondence

An *inter-model correspondence*, usually referred to (with a little abuse of terminology) also as *model correspondence*, is the expression of the relations holding between the "subparts" of two models. Such "subparts", according to the syntax we adopted for representing generic models, are identified by subgraphs of the considered models. A similar approach is used in [9]. A model correspondence is *Element-to-Element* if it puts in correspondence one element of a model with exactly one of the other model; there are, by the way, more complex cases in which we map single elements in the source model with subgraphs in the target one (*Element-to-Subgraph* correspondence) or, even, a subgraph in the source model with a subgraph in the target one (*Subgraph-to-Subgraph* correspondence). See figure 3.

Definition 4. *Given two models A and B ($A, B \in$ **MOD**) an inter-model correspondence is a relation $m \subseteq Sub(A) \times Sub(B)$. An (inter-)model mapping is a function fm: $Sub(A) \rightarrow Sub(B)$*

The above definitions are very general and capture a great variety of classes of correspondences: functional and non-functional correspondences, Element-to-Element correspondences, etc.

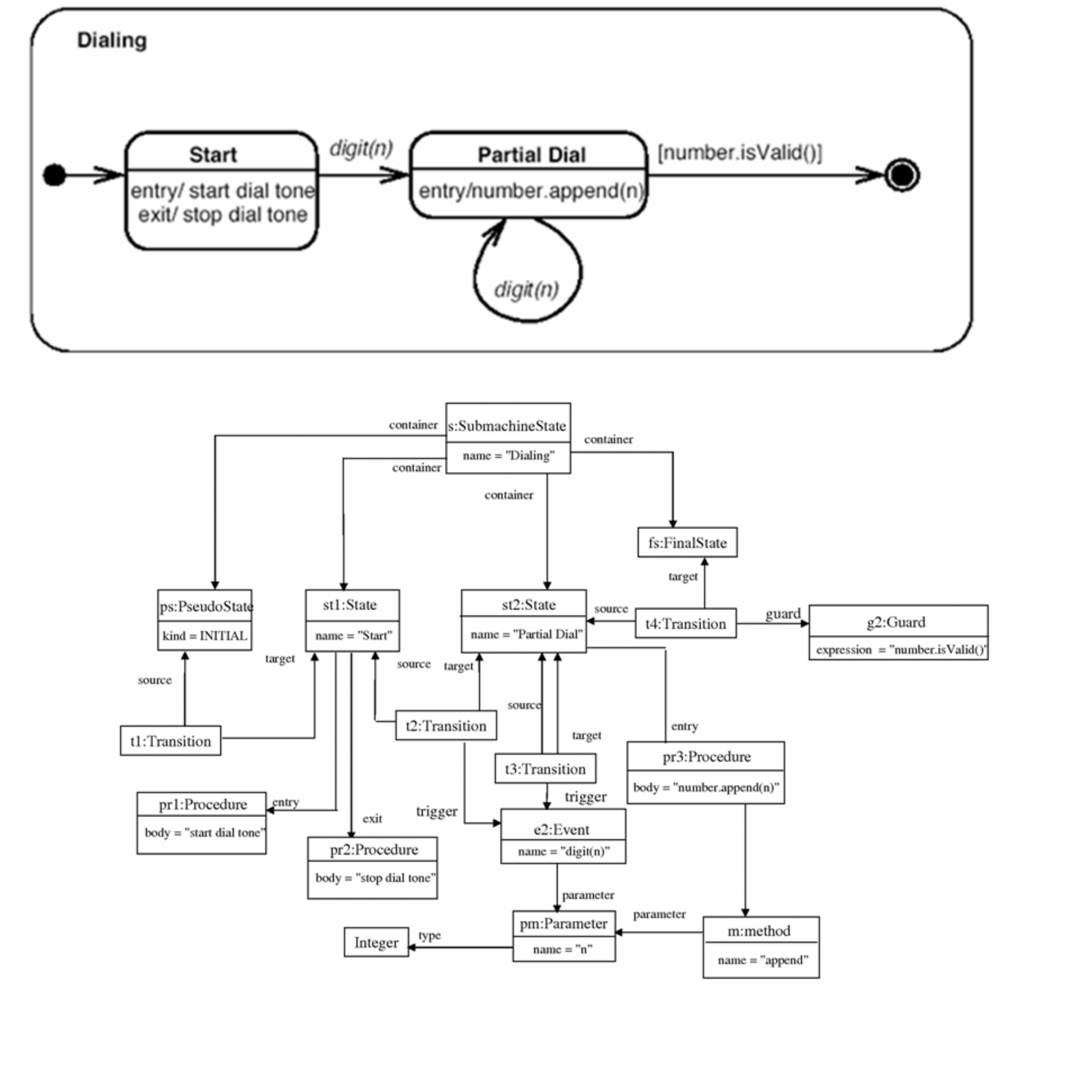

Fig. 2. UML State Diagram representation with LDMgraph formalism

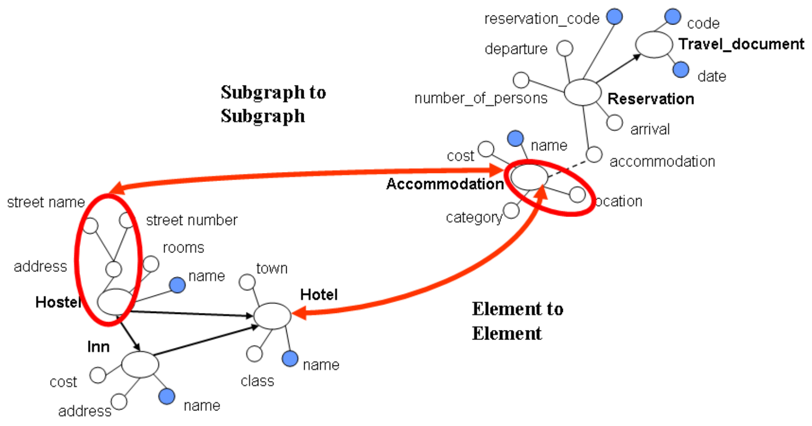

Fig. 3. Examples of *Element to Element* and *Subgraph to Subgraph* correspondance

Given two models A e B and a model correspondence m, three cases are possible:

- $\forall el \in EL_A$ exists $(x, y) \in m$ such that el is subgraph of x. In this case the model B is said to "totally cover" the model A via the correspondence relation m.
- $\exists el \in EL_A$ such that $\forall (x, y) \in m$, el is not a subgraph of x. In this case the model B is said to "partially cover" the model A via the correspondence relation m.
- $\forall el \in EL_A \nexists (x, y) \in m$ such that el is subgraph of x; this happens only when $m = \emptyset$.

The final considerations we want introduce concern the cardinality of the space of correspondences. It can be very large; in fact, given two models A and B, the space of correspondences $MSPACE_{AB}$ is constituted by the set of all relations $m \subseteq Sub(A) \times Sub(B)$, and thus $|MSPACE_{AB}| = 2^{|Sub(A) \times Sub(B)|}$.

Another common procedure is to enrich the correspondence with expressions taken from a given marking language, that will act as additional information in a subsequent step, the correspondence exploitation.

Definition 5. *Given two models A and B ($A, B \in \mathbf{MOD}$) and a marking language L_M, a marked correspondence is a relation $m \subseteq Sub(A) \times Sub(B) \times L_M$.*

A typical example could be the association to an edge of a number in the interval [0,1] to state the degree of similarity between the elements. By using as marking language $L_M = [0, 1]$, we can generate, for example, *(Employee, Person, 0.7)* or *(Employee, Student, 0.3)*.

5.1 Model correspondences and Semantics

In this section we want to introduce some simple cosiderations about the *semantics* of the models involved in a model correspondence.
Given an LDMGraph G, a *model interpretation of* G is a pair (I, Δ) where Δ is called the *domain of interpretation* or *universe of discourse* and I is a function such that $I(v) \subseteq \Delta$ for $v \in V_G$ and $I(e) \subseteq \Delta \times \Delta$ for $e \in E_G$. With a little extension we indicate with $I(G) = \bigcup_{el \in EL_G} I(el)$ to denote the interpretation of the model G.

Definition 6. *Given two models A, B and two model interpretations (I', Δ) and (I'', Δ) (note that that the domain of interpretation is the same) if it holds $I'(A) \subseteq I''(B)$ then the model A is* **semantically contained** *in the model B with respect to I', I''. The two models are* **semantically equivalent** *(w.r.t I', I'') if it holds the mutual containment.*

Given 2 models A and B, two model interpretations $(I', \Delta), (I'', \Delta)$ and a model correspondence $m \subseteq A \times B$, m is said to induce a **semantics-model containment** if for each (A', B') in m we have $I'(A') \subseteq I''(B')$.

6 Model Transformation

The scenario of a model transformation is the following: we have a source model A and a function t: $MOD \rightarrow MOD$ and we use it to obtain the target model B. So the definition of model transformation is fairly simple.

Definition 7. *Let A be a model and t: $MOD \rightarrow MOD$ be a function then t is a model transformation; A is called the source (input, starting) model and B = t(A) is called the target (output, transformed) model (of the application of the transformation t).*

The differences with a *model correspondence* (mapping) are the following:

– A model transformation is a function, a model correspondence can be a generic (non-functional) relation
– Domain and range of model correspondences and transformations are different (see definitions). Model correspondences define the relationships between subparts of two given models, while a model transformation takes as input a model and returns as output a model.

6.1 Model merging

The operation of merging raise when dealing with multiple input models. Fundamentally we consider model merging as a model transformation applied to multiple inputs.
Another intended meaning conveyed to the concept of model merging is related to the semantics preservation; we expect that the model result of merging semantically "contains" all the models given as inputs.
However we now intend to give only an intuition of merging functions.

Definition 8. *A* **multiple model transformation** *is a function t: $MOD^+ \rightarrow MOD$.*

We now propose a (not fully complete) definition of merging, letting undefined the concept of semantic preservation.

Definition 9. *A* **model merging** *is a multiple model transformation such that the semantics of the source models is preserved in the merged model.*

7 Model Correspondence Discovery

The process of (inter-model) correspondence discovery is the process that takes as inputs two models and returns as output a model correspondence.

Definition 10. *A (inter-model)c orrespondence (mapping) discovery is a function mdisc: $MOD \times MOD \rightarrow ICOR$ where $ICOR$ is the set of inter-model correspondences*

This definition, similarly to what has been done for the definition of model mappings and transformations, tries to be as general as possible. Once we have traced the "bounds" of the objects we are interested in, we are able to investigate and delineate several more specific subclasses.

For a detailed overview of types of correspondence discovery methods see [3].

8 Conclusions and future work

In this paper we tried to give a characterization of relevant terms in the domain of model mapping/transformation tasks, exploiting concepts and results from graph theory and from algebraic approaches to graph transformation. An abstract view of models in terms of graphs allows formal reasoning on their properties and on the properties of the transformation processes needed for their effective management. While several competing approaches to the definition of graph transformations exist, reasoning on abstract properties at an algebraic level provides several advantages. For example, it allows the study of the different forms of equivalence through general rather than ad hoc mechanisms, relying on a wealth of theoretical results. Moreover, it allows the exploitation of existing tools, without the need to implement ex novo complex pattern matching algorithms. Third, problems of model maintainance have been specifically addressed in these approaches, so that new problems can be attacked by adapting solutions for other related problems. Drawing on studies in these fields, we plan to arrive at a more refined characterization of *model transformation* and *model correspondence discovery techniques*, which will serve as a basis for the realisation of two reference architectures for model transformation (based on attributed graph transformations) and model correspondence discovery development.

References

1. P. Bottoni, P. Parisi-Presicce, and G. Taentzer. Specifying Coherent Refactoring of Software Artefacts with Distributed Graph Transformations. In P. v. Bommel, editor, *Transformation of Knowledge, Information, and Data: Theory and Applications*, pages 95–125. Idea Group Publishing, 2004.
2. Paolo Bottoni, Kathrin Hoffmann, Francesco Parisi Presicce, and Gabriele Taentzer. High-level replacement units and their termination properties. *Journal of Visual Languages & Computing*, 16:485–507, 2005.
3. Giorgos Stamou et al. Diana Maynard. D2.2.3: State of the art on current alignment techniques, 2004. Knowledge Web NoE Deliverable.

4. H. Ehrig, K. Ehrig, J. de Lara, G. Taentzer, D. Varró, and S. Varró-Gyapay. Termination criteria for model transformation. In *Proc. Fundamental Approaches to Software Engineering (FASE 2005)*, volume 3442 of *LNCS*, pages 49–63. Springer Verlag, 2005.
5. H. Ehrig, M. Gajewsky, and F. Parisi Presicce. *High-Level Replacement Systems with Applications to Algebraic Specifications and Petri Nets*, volume 3: Concurrency, Parallelism, and Distribution, chapter 4. to appear, handbook of graph grammars and computing by graph transformations edition, 1998.
6. H. Ehrig, A. Habel, H.-J. Kreowski, and F. Parisi-Presicce. Parallelism and concurrency in High Level Replacement Systems. *Math. Struc. in Comp. Science*, 1:361–404, 1991.
7. H. Ehrig and M. Löwe. Parallel and distributed derivations in the single pushout approach. *TCS*, 109:123 – 143, 1993. Tech. Rep. 91/01 Technical University Berlin.
8. H. Ehrig, M. Pfender, and H.J. Schneider. Graph grammars: an algebraic approach. In *14th Annual IEEE Symposium on Switching and Automata Theory*, pages 167–180. IEEE, 1973.
9. Giorgos Stamou et al. Franconi. D2.2.1: Specification of a common framework for characterizing alignment, 2004. Knowledge Web NoE Deliverable.
10. Jan Hendrik Hausmann, Reiko Heckel, and Marc Lohmann. Model-based discovery of web services. In *ICWS*, pages 324–331, 2004.
11. Jochen Malte Küster, Reiko Heckel, and Gregor Engels. Defining and validating transformations of uml models. In *HCC*, pages 145–152, 2003.
12. M. Löwe, M. Korff, and A. Wagner. Single-pushout transformation of attributed graphs: a link between graph grammars and abstract data types. In *Proc. of the SEMAGRAPH Symposium 1991*, pages 359–379. Technical Report 91-25, University of Nijmegen, 1991.
13. M. Löwe, M. Korff, and A. Wagner. An algebraic framework for the transformation of attributed graphs. In M.R. Sleep, M.J. Plasmeijer, and M.C. van Eekelen, editors, *Term Graph Rewriting: Theory and Practice*, chapter 14, pages 185–199. John Wiley & Sons Ltd, 1993.
14. T. Mens, S. Demeyer, and D. Janssens. Formalising Behaviour Preserving Program Transformations. In A. Corradini, H. Ehrig, H.-J. Kreowski, and G. Rozenberg, editors, *Proc. of ICGT'02*, volume 2505 of *LNCS*, pages 286–301. Springer, 2002.
15. Ed Seidewitz. What models mean. *IEEE Softw.*, 20(5):26–32, 2003.
16. G. Sunyé, D. Pollet, Y. LeTraon, and J.-M Jézéquel. Refactoring uml models. In *Proc. UML 2001*, volume LNCS 2185, pages 134–138. Springer-Verlag, 2001.
17. Gabriele Taentzer. Hierarchically Distributed Graph Transformation. In Janice E. Cuny, Hartmut Ehrig, Gregor Engels, and Grzegorz Rozenberg, editors, *Proc. 5th Int. Workshop on Graph Grammars and their Application to Computer Science*, volume LNCS 1073, pages 304–320. Springer-Verlag, 1996.
18. Ragnhild Van Der Straeten, Viviane Jonckers, and Tom Mens. A formal approach to model refactoring and model refinement. *Software Systems Modeling*, 2006.
19. Dániel Varró. Automated formal verification of visual modeling languages by model checking. *Software and System Modeling*, 3(2):85–113, 2004.
20. Wolfgang Wechler. *Universal Algebra for Computer Scientists.* Springer-Verlag, 1992.
21. Paul Ziemann, Karsten Hölscher, and Martin Gogolla. From UML models to graph transformation systems. In Mark Minas, editor, *Proceedings of the Workshop on Visual Languages and Formal Methods (VLFM 2004)*, volume 127(4) of *ENTCS*. Elsevier, 2005.