Electronic Communications of the EASST
Volume 30 (2010)


# International Colloquium on Graph and Model Transformation On the occasion of the 65th birthday of Hartmut Ehrig (GraMoT 2010)

## Graph Modelling and Transformation: Theory meets Practice

Karsten Ehrig and Claudia Ermel

21 pages

Guest Editors: Claudia Ermel, Hartmut Ehrig, Fernando Orejas, Gabriele Taentzer
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

# Graph Modelling and Transformation: Theory meets Practice

**Karsten Ehrig**[1] **and Claudia Ermel**[2]

[1] BAM Bundesanstalt für Materialforschung und -prüfung, Berlin, Germany
karsten.ehrig@bam.de

[2] Institut für Softwaretechnik und Theoretische Informatik
Technische Universität Berlin, Germany
claudia.ermel@tu-berlin.de

**Abstract:** In this paper, we focus on the role of graphs and graph transformation for four practical application areas from software system development. We present the typical problems in these areas and investigate how the respective systems are modelled by graphs and graph transformation. In particular, we are interested in the usefulness of theoretical graph transformation results and graph transformation tools in order to solve these problems. Finally, we characterize concepts and tool features which are still missing in practice to solve the presented and related problems even better.

**Keywords:** graph modelling, graph transformation, graph transformation tools

## 1 Introduction

Graphs are one of the key concepts for modelling. Since the early days of mankind, graphs are used to depict the relationship between two or more entities as abstractions of real world systems and processes. The visual nature of graphs makes them an intuitive language for human beings to think and discuss about partitioning systems into different components, and about processes of running systems which can be drawn as related but changing system state graphs. Throughout the history of software engineering, graph models have been used for software system design, such as entity-relationship diagrams for databases, class diagrams for static software structure, and the diagram types offered by the Unified Modeling Language (UML) [OMG07] to model different static and dynamic system aspects. Yet, when it came to programming, often enough a yawning gap opened between what the modellers meant when designing their graph models and what the programmers encoded using standard textual programming languages, where the graph models played the role of a rough guideline for programmers. Ambitious programming projects resulted in failure, went over their budgets or proved to be unstable over time.

Hence, the objective of model-driven development (MDD) is creating models closer to domain concepts rather than computing concepts [Béz05]. This means that for software developers the abstraction level is now raised. No longer do they need to worry about technical details and features of programming languages but can concentrate on more creative parts of software engineering: analysis, design and validation, all based on models. Sometimes, models are refined to a certain level of detail, and the code is written by hand in a separate step. Sometimes, code can be generated from models, ranging from system skeletons to complete, deployable products.

In all cases, the MDD perspective raises the importance of graph models and calls for rigorous methods to capture the semantics of graph models and their evolution over time [Eng00]. The fundamental notions behind graph models have been captured long ago by mathematical terms, thus allowing for rigorous reasoning at model level. Yet, experience shows that many problems in using formal methods in software development arise because the formal model and the problem domain are too far apart. Since any software system is situated in a particular social context, this context (domain) should be represented also in models based on formal notations. Here, again, graph models with their visual nature are a good candidate for uniting the domain-specific and the formal aspects of real-world problems. Domain specific languages based on graphs may use a graphical concrete syntax with adequate intuitive symbols which are manipulated adequately to model dynamic system aspects. Thus, the system behaviour may be animated in a domain-specific visualization to validate system properties by domain experts.

Since real world systems evolve, their models need to model evolution as well. Algebraic graph transformation is a formally defined calculus based on graphs and graph transformation rules [EEPT06]. For ages, rules have proven to be extremely useful for describing computations by local transformations. Areas like language definition, logic, functional programming, algebraic specification, term rewriting and expert systems have rules as key concepts. Graph transformation, also known as graph rewriting or graph reduction, combines the potential and advantages of both graphs and rules into a single computational paradigm.

In this paper, we summarize a few selected case studies from recent literature which have been modelled by graphs and algebraic graph transformation (reviewed in Section 2). In particular, we focus on four case studies from different application areas: a medical information system (Section 3), a model transformation between two different modelling notations (Section 4), a metabolic pathway analysis (Section 5), and a self-healing system (Section 6). For each application area, we ask the following questions:

1. What are typical problems in this area?
2. How can they be modelled by graphs or graph transformation?
3. What kind of graph transformation results can be applied to solve these problems?
4. What are missing graph modelling and transformation concepts and results?

In the evaluation (Section 7), we summarize the experiences gained from the case studies and state what kinds of concepts and results we find still missing.

## 2 Algebraic Graph Transformation: Background

For nearly 40 years, graph transformation has been studied in a variety of approaches, motivated by application domains such as pattern recognition, semantics of programming and visual modelling languages, specification of distributed systems etc. [EEKR99, EKMR99, BTMS99].

A detailed presentation of different graph transformation approaches, is given in volume 1 of the *Handbook of Graph Grammars and Computing by Graph Transformation* [Roz97]. The *algebraic approach* is based on pushout constructions, where pushouts are used to model the

gluing of graphs. In fact, there are two main variants of the algebraic approach, the double and the single pushout approach. The double pushout (DPO) approach [EEPT06], is the formal basis for visual modelling of behavioural models and model transformations considered in this article. The DPO approach is based on category theory: a graph transformation rule is a pair of morphisms in the category of graphs with total graph morphisms as arrows: $r = (L \leftarrow K \rightarrow R)$ where $K \rightarrow L$ is injective. Graph $K$ is called *gluing graph*. Another graph morphism $m\colon L \rightarrow G$ models an occurrence of $L$ in $G$ and is called a *match*. Intuitively, this means that $L$ is a subgraph that is matched to $G$, and after a match is found, the rule can be applied.

A *direct transformation* or application of rule $r$ to graph $G$ is defined by two pushout diagrams (see the diagram to the right). Applying the rule, $m(L)$ is replaced with $m^*(R)$ in graph $G$, leading to the transformed graph $H$. A *graph transformation*, or, more precisely, a graph transformation sequence, consists of zero or more direct transformations, written $G_0 \stackrel{*}{\Longrightarrow} G_n$. A set of graph rules is called *graph transformation system*. A *type graph* defines a set of types which can be used to assign a type to the nodes and edges of a graph. The typing itself is done by a graph morphism from the graph to the type graph. A *typed graph transformation system* $GTS = (TG, P)$ consists of a type graph $TG$ and a set $P$ of typed graph rules. A *(typed) graph grammar* $GG = (GTS, S)$ consists of a (typed) graph transformation system $GTS$ and a (typed) start graph $S$. The *(typed) graph language* $L$ of $GG$ is defined by $L = \{G \mid \exists \text{ (typed) graph transformation } S \stackrel{*}{\Longrightarrow} G\}$. The key idea of *attributed* graph transformation is to model graphs with node and edge attributes, i.e. an attributed graph is a pair $AG = (G, A)$ of a graph $G$ and a data type algebra $A$. *Typed attributed* graph transformation, combining process and data modelling proved to be well-suited to define and analyse visual models and model transformations [EEPT06, MVVK05].

A variety of tools for graph transformation exist [TEG+05] to be used as transformation engine and for analysis purposes, to reason about issues such as conflicts and dependencies of actions as well as consistency of object structures.

## 3 Case Study 1: Medical Information System

**Problem**

Information systems are very common nowadays in almost all common application areas of software systems. In health care, data from different domains like admission, physical examination, medical record archive, etc. have to be coordinated and presented to the employees. Data manipulations, like the admission of a new patient, have to be supported intuitively.

**Aim of the Model**

An interactive visual application with a suitable graphical user interface shall be generated from a suitable model. Instead of complex textual data, visual symbols shall be used to support the necessary information system operations. The operations shall be modelled in a precise, unambiguous way.

**Technique to solve the problem / realize the aim**

We use typed, attributed graphs to model the abstract syntax of the information systems, and graph transformation rules on the abstract syntax model to define the operations to be performed by the clinical staff. Moreover, we combine the abstract syntax elements with concrete syntax symbols to visualize graphs in an adequate, domain-specific way. Constraints and application conditions are used to check the consistency of the model and the operations to be performed. From this model, the interface and operation code allowing the users to operate on the information system visually is generated automatically.

**Overview of the model**

Figure 1 shows icons for patients, beds, rooms, admission and discharge (from left to right) used in our information system.


Figure 1: Graphical Symbols for Medical Information System

In Figure 2, the current ward patient allocation diagram shows bed icons inside the room icons to represent the number of available beds in the ward rooms.

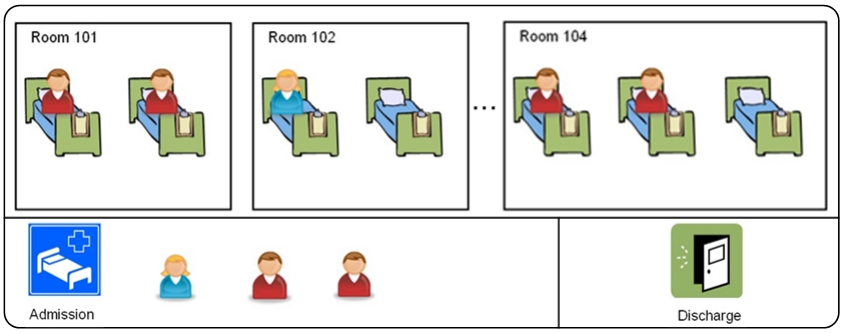



Figure 2: Sample User Interface Diagram for Medical Information System

A patient icon is connected with a bed if occupied, otherwise the bed is left empty. Patients currently not associated with a bed are shown next to the admission symbol. This requires a user action. Dragging an *female patient* symbol onto a *free bed* symbol evokes rule *Admission* (Figure 3).

Applying this rule, the user of the information system assigns a female patient to a bed and a room, unless there are male patients in the same room (modelled by a NAC). With a visual rule editor, the information system designer may define new rules and user policies according to the needs and standards of the hospital.

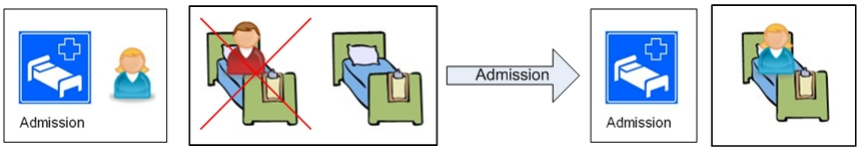



Figure 3: Sample Rule *Admission*

**Tool Support**

TIGER 2 [BEEH09] is an generator of modeling tool environments for visual domain specific languages. In the modelling environment, a set of graph transformation rules called editing rules define the editing commands of the generated visual editor, i.e. the model syntax; on the other hand, a set of simulation rules may describe a model's operational semantics.

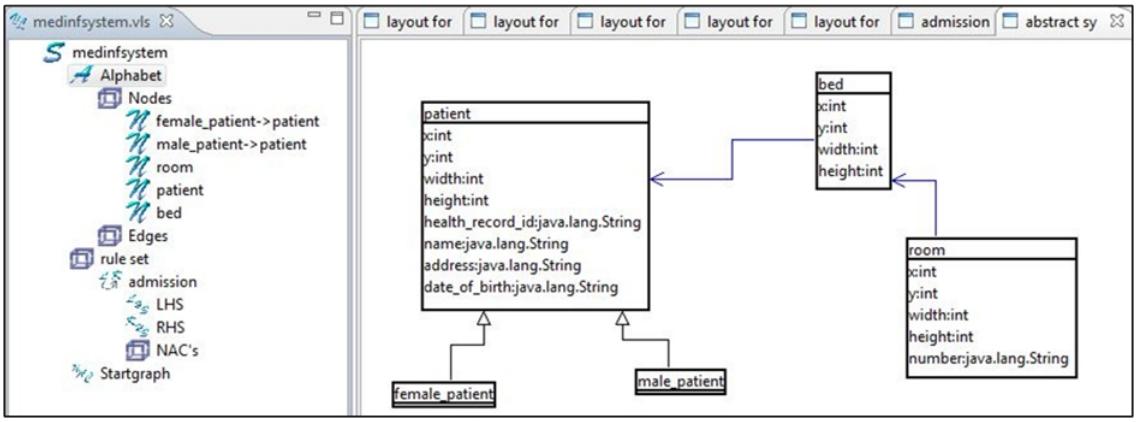



Figure 4: Abstract Syntax definition in TIGER 2

Figure 4 shows the (simplified) abstract syntax of the case study modelled in TIGER 2. A *patient* is associated with a *bed* located in a *room* of the ward numbered with the attribute *number* of data type *String* to allow for combinations of letters and numbers (e.g. 'room A15'). Node *patient* is an abstract node, specialized to nodes *female_patient* and *male_patient*. The *patient* attribute *health_record_id* of is used for unique identification of the current health record in the system database. One patient may acquire more than one health record ids for different admissions. The attributes *x*, *y*, *width*, and *height* are used for icon visualization.

**Related Work**

Starting with an EMF domain model, the Graphical Modeling Framework (GMF) [GMF07] provides a code generation facility for a graphical editor with basic editor operations for inserting graphical objects and links between them. Apart from GMF [GMF07], also the TOPCASED modeler generator of the OPENEMBEDD [Ope09] MDE platform provides graphical patterns for common parts of user specific EMF domain models and thus allows to easily create a basic graphical editor, visualizing mainly the abstract model syntax as graph-like diagrams with nodes

and edges. More sophisticated tools generating graph-based modelling environments that can be customized to various domains are e.g. Metaedit+ [TR03], the Generic Modeling Environment GME [AKL03] and DiaGen [Min07], a diagram editor generator based on graph transformation.

**Unsolved Problems**
Graph-based modelling environments need to integrate various domain specific editors and views for defining e.g. simulations and model transformations. All views have to be interconnected and customized to the domain. Up to now, such *multi-view editors* cannot be generated automatically from domain models by generators like GMF. We are convinced that a combination of EMF-based modeling tools [EMF09] and graph transformation tools [Tae06] provide a solid basis to define complex operations for editing, simulation, and model transformation of domain specific languages based on a well-defined theoretical background [BET08]. Up to now, a comprehensive generation framework combining graph transformation, EMF-based meta-modeling and for the generation of customized visual modelling environments has not yet been implemented.

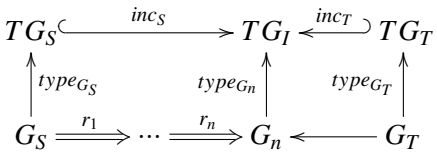
## 4 Case Study 2: Business Process Model Transformation

**Problem**
The Business Process Modelling Notation (BPMN) [Whi04] is a graph-oriented language in which control and action nodes can be connected almost arbitrarily. It defines a Business Process Diagram (BPD), which is a kind of flowchart incorporating constructs tailored to business process modelling, such as AND-split, AND-join, XOR-split, XOR-join. It is supported by various modelling tools but so far no systems can directly execute BPMN models. The Business Process Execution Language for Web Services (BPEL) [IBM03], on the other hand, is a mainly block-structured language. BPEL is emerging as a de-facto standard for implementing business processes on top of web services technology. Numerous platforms support the execution of BPEL processes.

**Aim of the Model**
The aim of this case study is to define the BPMN2BPEL model transformation at an adequate abstraction level. A challenge in formalizing the particular model transformation is the translation of BPMN *And* and *Xor* constructs to the corresponding BPEL language elements *Flow* and *Switch*. Translating those constructs with ordinary graph transformation rules requires a complex control structure for guidance. We aim for an intuitive, visual description of the model transformation where arbitrary many branches of *And* and *Xor* constructs can be treated in parallel.

**Technique to solve the problem / realize the aim**
We use typed, attributed graph transformation based on an integrated type graph $TG_I$. This type graph consists of the type graphs for the source and target language, and, additionally, reference nodes with arcs mapping source elements to target elements. We express model transformations directly by $TG_I$-typed graph transformation rules $L \leftarrow K \rightarrow R$ where $L$ basically represents source model elements, and $R$ represents the corresponding generated target model elements. The model

transformation starts with graph $G_S$ typed over $TG_S$. As $TG_S$ is a subgraph of $TG_I$, $G_S$ is also typed over $TG_I$. During the model transformation process, the intermediate graphs $G_S = G_1, .., G_n$ are all typed over $TG_I$. To delete all items in $G_n$ which are not typed over $TG_T$, we can construct a restriction (a pullback in the category **Graphs**), which deletes all those items in one step. In addition to normal graph transformation rules, we also use rule schemes to express parallel transformation of arbitrary many similar model element patterns. The application of rule schemes is defined by the concept of amalgamated graph transformation [BFH87].




**Overview of the Model Transformation**

The complete model transformation case study is described in [BEE+10]. The type graph integrating the BPMN source model (left-hand part), the reference part connecting source and target model (the node type F2ARef and its adjacent edge types bpmn and bpel), and the BPEL target model (right-hand part) is shown in Figure 5.

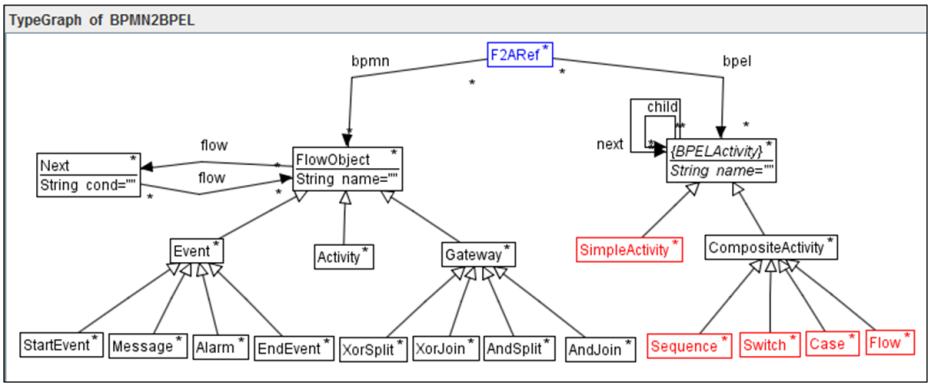



Figure 5: BPMN2BPEL type graph

As an example we consider a BPMN diagram which models a person's interaction with an ATM (see Figure 6 where the concrete and abstract syntax of the diagram are depicted). In the upper part, the ATM machine accepts and holds the card of the person while simultaneously contacting the bank for the account information. (The language elements AndSplit and AndJoin are used to model parallel actions.) Afterwards, the display prompts the user for the PIN. Depending on the user's input there are three alternative actions possible: (1) the user enters the correct PIN and can withdraw money, (2) a wrong PIN is entered – a message is displayed, (3) the operation is aborted – an alarm signal is given.

We give one example for a model transformation rule scheme (in abstract syntax) to translate Xor constructs. All other rules and rule schemes can be found in [BEE+10]. An Xor construct (a number of branches surrounded by an XorSplit and XorJoin element) is translated to a *Switch* container node which contains a child for each branch emerging from the XorSplit. Since the number of branches can be arbitrary, a normal graph transformation rule or any finite number

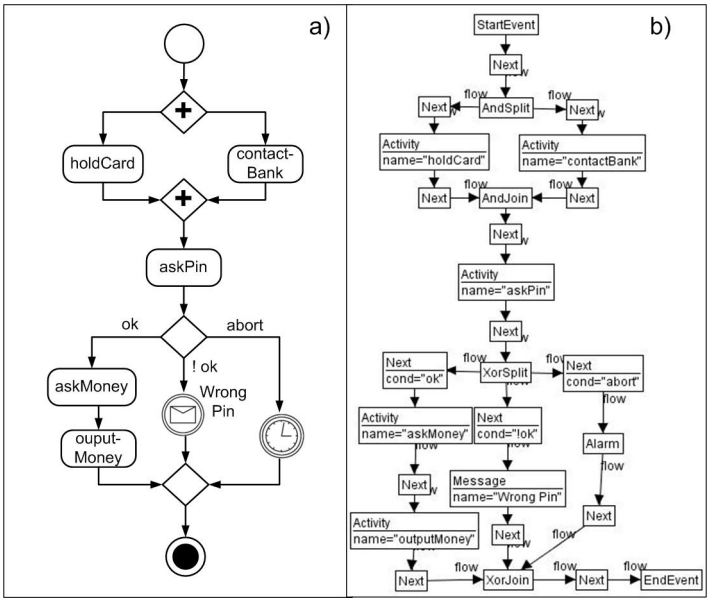



Figure 6: ATM machine in BPMN in concrete syntax (a) and abstract syntax (b)

of rules would not be sufficient to express this situation. Therefore, we here use *amalgamated graph transformation*, a technique to specify *forall*-operations on recurring model patterns (e.g. for each branch in an And construct). A multi-rule scheme contains a fixed *kernel rule* part and the recurring model pattern (called *multi-rule*). The kernel rule part defines the elements in the graph which are common to all recurring model patterns (e.g. the XorSplit and XorJoin nodes that surround all branches). An *amalgamated rule*, induced by such a scheme, is a kind of parallel rule operating on all recurring model patterns in parallel but synchronized by the kernel rule part. Applying the amalgamated rule to a graph, it modifies all recurring matches of the model pattern, which overlap in the match of the kernel rule in one step.

We model a multi-rule scheme as a rule embedding of the kernel rule part into the multi-rule, which contains in addition to the kernel rule part the recurring model pattern. The upper part of Figure 7 shows the kernel rule part, where one branch surrounded by an XorSplit and XorJoin is translated to a BPEL Switch node with one Case branch where the condition in the Next node is translated to a Case distinction. The multi-rule for processing And constructs is shown in the bottom part of Figure 7. It extends the kernel rule by one more branch, which comprises the recurring model pattern, and translates it accordingly. The rule embedding from the kernel rule to the multi rule is indicated in Figure 7 by corresponding numbers of some of the graph objects. For applying a multi-rule scheme, first, a match of the kernel rule is selected. Then, copies of the multi-rule are constructed, one for each new match of a multi-rule in the current host graph that overlaps with the match of the kernel rule. At last, all multi rule copies are glued at their corresponding kernel rule objects which leads to a new rule, the *amalgamated rule*. The application of the amalgamated rule is called *amalgamated graph transformation*.

The application of the multi-rule scheme CreateSwitch in Figure 7 to the ATM graph in Figure 6 yields the amalgamated rule in Figure 8 where the kernel rule is glued with two multi-rule copies

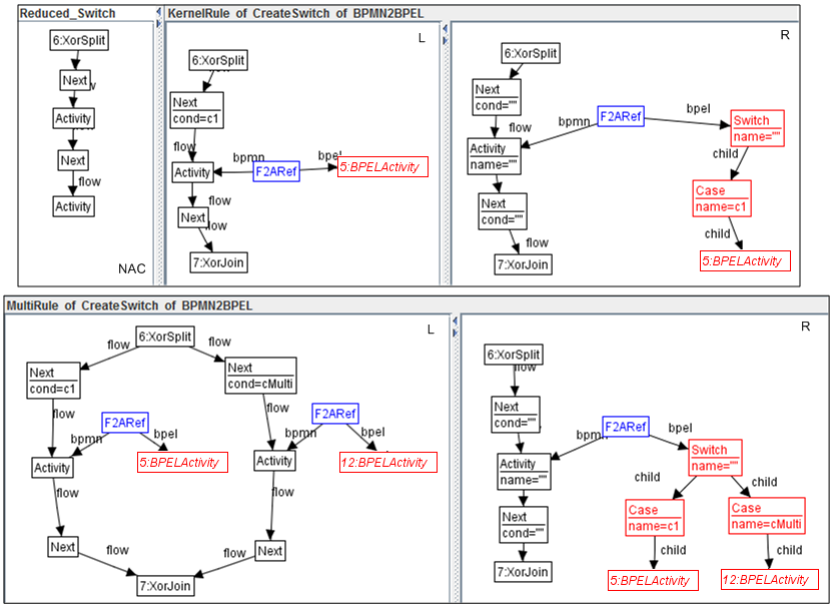



Figure 7: Multi-rule scheme *CreateSwitch*

(since we have three branches in Figure 6 between the XorSplit and the XorJoin). The amalgamated rule in Figure 8 is then used to translate the three branches in one step by applying it to the ATM graph in Figure 6.

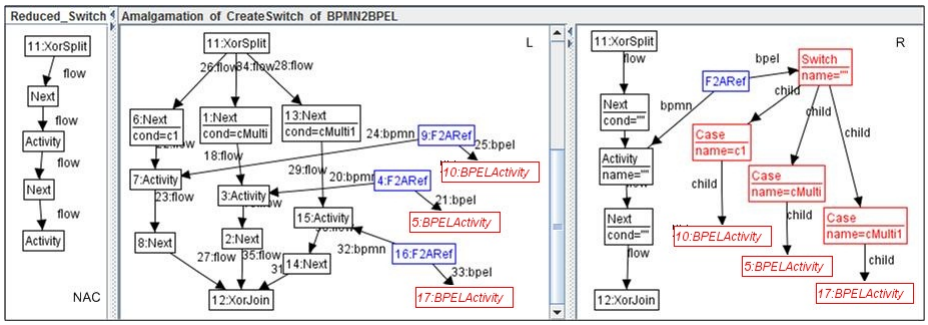



Figure 8: Amalgamated rule of scheme *CreateSwitch* constructed for the ATM model in Figure 6

For this case study, a theoretical result [GEH10] is applied which allows us to show parallel independence of amalgamated graph transformations by analyzing the underlying multi-rules. Hence, we may translate the And construct and the Xor construct using amalgamated graph transformation in arbitrary order. After applying our transformation rules and schemes starting with the ATM model in Figure 6, we get the resulting integrated graph shown in Figure 9 (b). The abstract syntax of the BPEL expression is the red tree with root node Sequence. The concrete syntax of the BPEL model corresponding to this tree is shown in Figure 9 (a).

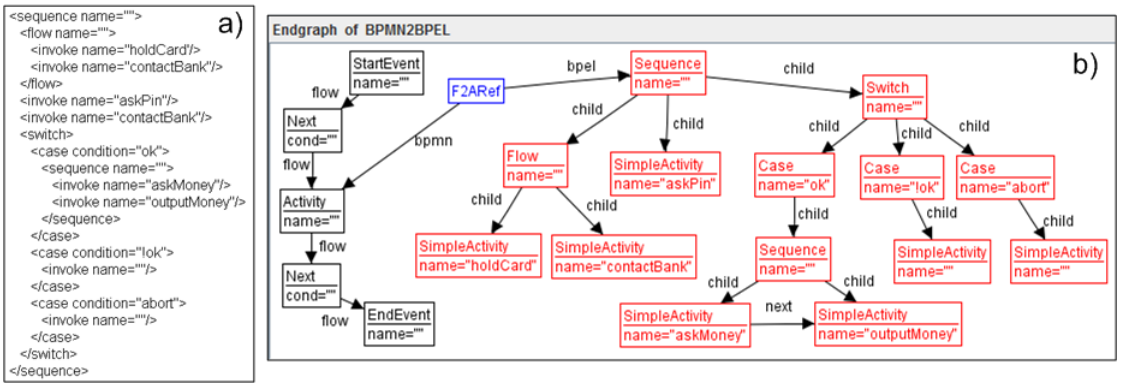



Figure 9: ATM machine after transformation: (a) in concrete BPEL syntax, (b) in abstract syntax

**Tool Support**

We implemented the case study in our tool AGG [AGG09, BEL+10], supporting the definition of type graphs, typed attributed graph rules and constraints. AGG has been extended recently by support for defining and applying amalgamated graph transformation. All screenshots in this section are taken from the AGG editors for rules and interaction schemes. Moreover, AGG supports verification of model transformations w.r.t. termination and confluence (functional behaviour).

**Related Work**

A related model transformation approach based on graph transformation are triple graph grammars (TGGs) [Sch94] which transform pairs of related models simultaneously while maintaining their consistency. TGGs generate languages of *triple graphs*, consisting of a source graph $G^S$ and a target graph $G^T$, together with a correspondence graph $G^C$ "between" them. A triple graph is typed by a meta-model triple which contains the source and target meta-models, and declares the types of mappings between the elements of both languages. A *triple rule* $tr$ consists of triple graphs $L = (SL \leftarrow CL \rightarrow TL)$ and $R = (SR \leftarrow CR \rightarrow TR)$, and an injective triple graph morphism $tr = (s,c,t) : L \rightarrow R$, representing a non-deleting rule which adds target elements. Further graph transformation tools tuned for domain-specific model transformations are VIATRA2 [BNS+05] and the Graph Rewriting and Transformation Language (GReAT) [SAL+03]. A tool that also supports amalgamated graph transformation is AToM$^3$ [LVA04] where the technique is used for model simulation [LETE04].

**Unsolved Problems**

An open problem is the semantical correctness of model transformations. In order to be semantically correct, a model transformation should lead to target models which behave equivalently w.r.t. the corresponding source models. This is an important property of e.g. code generators for behavioural models. In the case that a model is more abstract than the code, semantical properties are defined explicitly, and it has to be shown that these properties are fulfilled by the respective pairs of source and target models.

# 5 Case Study 3: Metabolic Pathway Analysis

**Problem**
Metabolic pathway analysis is one of the tools in biology and medicine in order to understand chemical reaction cycles in living cells. The problem is that often, reactions are analysed at the level of structural formulae only, thus summarising the number of atoms of certain types in a compound without keeping track of their identity.

**Aim of the Model**
This case study [EHL06] aims at understanding chemical reactions at the level of individual atoms or component molecules. In particular, we are interested in the analysis of causal dependencies between biochemical reactions. Given a metabolic pathway (a sequence of reactions) we would like to be able to trace the history of particular atoms or molecules. This is relevant, for example, when trying to anticipate the outcome of experiments using radioactive isotopes of such atoms. Such questions have been crucial to the detailed understanding of the nature of reactions like the citric acid cycle.

**Techniques used to solve the problem / realize the aim**
Biological systems and chemical reactions are characterized by their inherent concurrency, allowing reactions to take place simultaneously as long as they involve different resources and to keep track of causal dependencies and conflicts between them. Graph transformation systems provide concurrency concepts which are suitable to be applied in this area. For modeling the metabolic pathway, we propose a new hypergraph model for chemical compounds which refines the classical representation in terms of structural formulae in two different ways.

- Our representation keeps track of the identity of atoms or molecular components by means of the identities of hyperedges. In contrast, when writing down chemical reactions with structural formulae, the identities of the reacting atoms are not explicitly represented in the notation. In situations where several atoms of the same element are involved, this lack of information leads to ambiguity as to where a new atom is placed in the resulting molecule. Our graph transformation-based model allows to track atom identities by graph homomorphisms between the graphs representing the compounds before and after the reaction.

- Modelling atoms as hyperedges, each connected to an ordered sequence of nodes, the relative spatial orientation of different molecular components is recorded through the ordering of the nodes connected to a hyperedge.

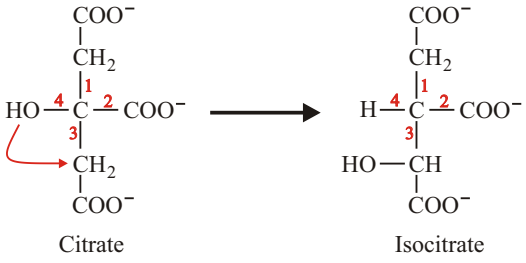
Using this model we are able to trace the dependencies between different steps in the reaction based on individual atoms and their spatial arrangement.

Formally, given a ranked set of labels $\mathscr{A} = (\mathscr{A}_n)_{n\in\mathbb{N}}$, an $\mathscr{A}$-labelled hypergraph $(V,E,s,l)$ consists of a set $V$ of vertices, a set $E$ of edges, a function $s : E \to V^*$ assigning each edge a sequence of vertices in $V$, and an edge-labelling function $l : E \to \mathscr{A}$ such that, if $length(s(e)) = n$ then $l(e) = A$ for $A \in \mathscr{A}_n$, i.e., the rank of the labels determines the number of nodes the edge is attached to. A morphism of hypergraphs is a pair of functions $\phi_V : V_1 \to V_2$ and $\phi_E : E_1 \to E_2$ that

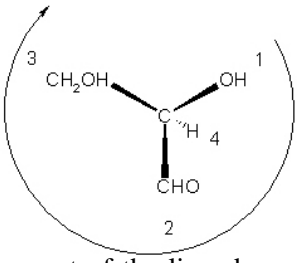
preserve labels and assignments of nodes, that is, $l_2 \circ \phi_E = l_1$ and $\phi_V^* \circ s_1 = s_2 \circ \phi_E$. A morphism thus has to respect the atom represented by an edge and also its chemical valence (number of bonds). Labelled hypergraphs can be considered as hierarchical graph structures. As shown by Löwe [Löw93], pushouts can be computed elementwise for all hierarchical graph structures and therefore the standard graph transformation approaches can be applied.

**Overview of the Model**

We consider as an example the citric acid cycle, a classical, but non-trivial reaction for energy utilisation in living cells [ZPV95]. Our approach supports a molecular analysis of the cycle, tracing the flow of individual carbon atoms based on a simulation. This cycle is a series of chemical reactions of central importance in all living cells that utilise oxygen as part of cellular respiration. Starting with *acetyl-CoA*, one of the resulting products of the chemical conversion of carbohydrates, fats and proteins, the citric acid cycle produces fast usable energy in the form of *NADH*, *GTP*, and $FADH_2$ which are precursors of the well known *adenosine-tri-phosphate (ATP)*. The diagram to the right shows reaction 2 of the citric acid cycle. The input agent of reaction 2, *citrate*, has two $CH_2COO^-$ groups, one on the top and one on the bottom. To fit into the enzyme *aconitase* catalysing reaction 2, only the $CH_2COO^-$ group marked with 3 is able to fit into the enzyme due to 3-dimensional spatial relations.




In our hypergraph model, we interpret the hyperedges as atoms and the nodes as bonds between them. The string $s(e)$ of vertices incident to an edge $e \in E$ gives the specific order of the bonds to other atoms, coding also their spatial configuration, as we will see. As ranked set of labels, we use $\mathscr{A}_1 = \{\mathrm{H}, \mathrm{CH_3}, \mathrm{OH}, \ldots\}$, $\mathscr{A}_2 = \{\mathrm{O}, \mathrm{CH_2}, \mathrm{S}, \ldots\}$, $\mathscr{A}_3 = \{\mathrm{CH}, \mathrm{N}, \ldots\}$, $\mathscr{A}_4 = \{\mathrm{C}, \mathrm{S}, \ldots\}, \ldots$ to denote elements of the periodic system or entire chemical groups. The rank of a label models the valence of an atom. For instance, a carbon atom with $l(e) = \mathrm{C}$ always has $s(e) = v_1 v_2 v_3 v_4$, a word of length 4. Hence, we define C as a label of rank 4. For elements with more than one possible valence (e.g. sulphur), the corresponding label can belong to several of the sets $\mathscr{A}_n$. Given an organic molecule, we represent the 3-dimensional configuration of the ligands of a C atom as a hypergraph by relating it to D-glyceraldehyde, one of the simplest chiral organic compounds. We impose a numbering on the ligands of a carbon atom such that a substitution of ligand 1 by OH, ligand 2 by CHO, ligand 3 by $CH_2OH$, and ligand 4 by H would result in D-glyceraldehyde. This convention defines the spatial arrangement of the ligands unambiguously. Substitution of ligands may change the angles between the ligands, and they often differ from the regular tetrahedral angle of $109°28'$, but the so called *angle strain* [Mos96] does not affect the uniqueness of the molecule represented by our notation.



As example, Figure 10 shows the representation of the prochiral molecule *citrate* as a hypergraph, where

$$V = \{v_1, v_2, \ldots, v_6\}, E = \{e_1, e_2, \ldots, e_7\},$$

$$s(e_1) = v_1, s(e_2) = v_1v_2, s(e_3) = v_3, s(e_4) = v_2v_3v_4v_5, s(e_5) = v_4, s(e_6) = v_5v_6, s(e_7) = v_6$$

$$l(e_1) = \mathrm{COO}^-, l(e_2) = \mathrm{CH_2}, l(e_3) = \mathrm{OH}, l(e_4) = \mathrm{C}, l(e_5) = \mathrm{COO}^-, l(e_6) = \mathrm{CH_2}, l(e_7) = \mathrm{COO}^-$$

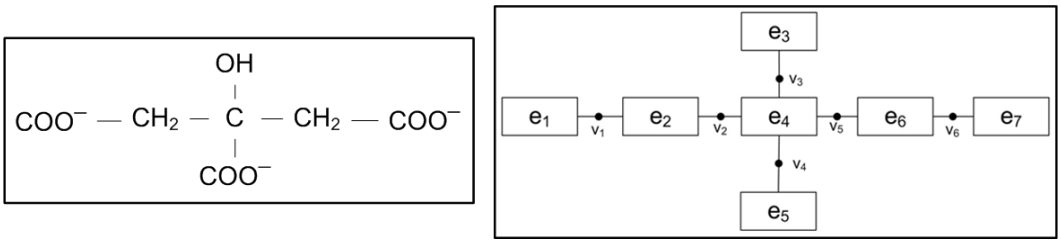



Figure 10: Structural formula (left) and hypergraph representation (right) of citrate.

**Tool Support**

We provide an encoding of the model in terms of attributed bipartite graphs that can be implemented in the AGG system for simulation and analysis [Tae04, AGG09].

Figure 11 shows reaction 2 of the citric acid cycle modelled in AGG. The enzyme *aconitase* accepts only the source agent *citrate* with the indicated *o* edge attribute order of the *1:C* atom in the left-hand side of Figure 11. In this reaction the OH group of the *1:C* atom is exchanged with the OH group of the *3:C* atom. This leads to the new agent *isocitrate*.

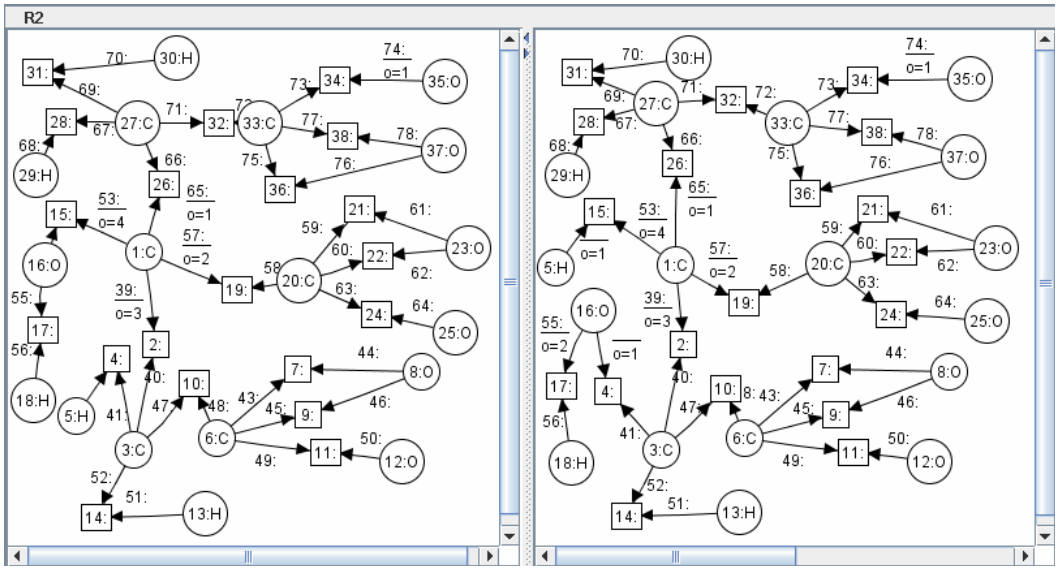



Figure 11: Reaction 2 of the citric acid cycle in AGG.

**Related Work**
The use of graph transformation for biological systems has a long history (see [RV05]), but early applications were mostly devoted to the field of morphogenesis. Our approach focuses on biochemistry, a field which gained much importance in the last decades because of the growth of biotechnology. Providing automated assistance for analyzing biochemical reactions can help in understanding the principles which govern the processes in living cells.

**Unsolved Problems**
The citric acid cycle is a very common cycle for energy utilization in living cells. However, biological systems are very complex and hard to understand, so most of the biological pathways are still not completely understood. For analyzing more complex pathways, big computer clusters are needed. Modelling with graph transformations might produce an overhead of data structures for the internal representation and computation with graphs. In general, the graph transformation problem is NP-complete. Putting several reactions together, the system might be unsolvable in a usable time frame. Recently, more focus has been given to the comparison of graph transformation tools with respect to performance (memory usage and efficiency). Starting with Varró's benchmark study [VSV05], a transformation tool contest[1] is held regularly nowadays where scalability and performance of graph-based transformation tools are important issues (see also [RV10]).

## 6 Case Study 4: Self-Healing Automated Traffic-Light

**Problem**
Self-healing (SH-)systems are characterized by an automatic discovery of system failures, and techniques how to recover from these situations. The problem is that failures can occur at any time during system operation. It is very important for such systems that recovery actions can always be applied after a failure has occurred and that they always lead to a system that works as expected.

**Aim of the Model**
The aim of our model is to verify that SH-systems have certain self-healing properties. For an SH-system, we distinguish *reachable*, *failure* and *normal* states (depending on which sets of constraints they fulfill), where reachable states split into normal and failure states. In particular, we call an SH-system is *self-healing* if each system state considered as *failure* state can be repaired, i.e. the system state after the repair action is considered as *normal* state. Furthermore, we want to ensure certain liveness properties of SH-systems. We call an SH-system *deadlock-free* if no reachable system state is a deadlock. A stronger liveness property is *strong cyclicity*, meaning that each pair of reachable states can be reached from each other.

**Technique to solve the problem / realize the aim**
In this case study, we model SH-systems by typed attributed graph transformation systems en-

[1] Case studies and solutions of the last three years' contests are available at http://www.planet-research20.org/ttc2010/

riched with graph constraints expressing their operational properties. We make use of theoretical results, i.e. sufficient static conditions for self-healing properties, deadlock-freeness and liveness of SH-systems.

**Overview of the Model**

The complete case study is given in [EER+10]. We model an automated Traffic Light System (TLS). The traffic light technology is based upon electromagnetic sensors buried some centimeters underneath the asphalt of car lanes. The sensors register traffic data and send them to other system components. The TLS is connected to cameras which record videos of the violations and automatically send them to the center of operations. In addition to the normal behavior, we may have failures caused by a loss of signals between a traffic light or a camera and the supervisor component. For each of the failures there are corresponding repair actions which can be applied after monitoring the failures during run-time.

We define the Traffic Light SH-system *TLS* by a type graph $TG$, an initial state, a set of normal rules $R_{norm}$ (modelling the ideal behaviour), a set of failure rules $R_{fail}$ modelling failures, a set of repair rules $R_{repair}$, which are the inverse repair rules, and sets of constraints that characterize properties of states being either consistent or failure states.

In our example, we model a single traffic crossing with two traffic lights in directions north-south and east-west. In the initial state (see Figure 12), both traffic lights are red and there are no cars at the crossing. The TL nodes represent the traffic lights, connected to a crossing supervisor component, and to cameras which are currently not in use (onCamera=false). The infraction attribute becomes true in the case that a car runs a red light.

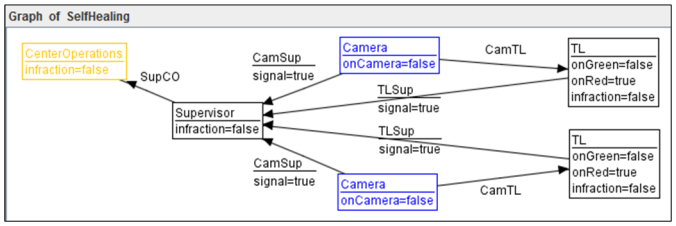



Figure 12: TLS initial state $G_{init}$

Normal rules $R_{norm}$ model the behaviour of cars arriving at the crossing and leaving it, as well as cars running a red light and being filmed by a camera. Failure rules $R_{env}$ (applied from the environment) model the loss of a signal of either a traffic light (in this case the signal attribute of a TLSup edge changes to false), or of a camera (here, the signal attribute of a CamSup becomes false). Repair rules $R_{rpr}$ model the recovery from the respective signal loss. In [EER+10], we formalize operational properties, including self-healing and deadlock-freeness and provide static conditions for them based on rule set analysis.

An SH-System SHS is called *self-healing*, if each failure state can be repaired, i.e. $\forall G_{init} \Rightarrow^* G$ via $(R_{norm} \cup R_{env})$ with $G \in Fail(SHS)$ $\exists$ $G \Rightarrow^+ G'$ via $R_{rpr}$ with $G' \in Norm(SHS)$. SHS is called *deadlock-free*, if no reachable state is a deadlock, i.e. $\forall G_0 \in Reach(SHS)$ $\exists$ $G_0 \stackrel{p}{\Longrightarrow} G_1$ via $p \in R_{norm} \cup R_{env} \cup R_{rpr}$. In particular, SHS is *normally deadlock-free*, if no state reachable via normal rules is a (normal) deadlock, i.e. $\forall G_{init} \Rightarrow^* G_0$ via $R_{norm}$ $\exists$ $G_0 \stackrel{p}{\Longrightarrow} G_1$ via $p \in R_{norm}$.

SHS is *strongly cyclic*, if each pair of reachable states can be reached from each other, i.e. $\forall G_0, G_1 \in Reach(SHS)\ \exists\ G_0 \Rightarrow^* G_1$ via $R_{norm} \cup R_{env} \cup R_{rpr}$.

For the analysis of SH-systems, we have the following results concerning self-healing properties and deadlock-freeness: An SH-System is self-healing, if it has the following three properties: 1) the initial state is normal and all normal rules preserve normal states, 2) each pair $(p,q) \in R_{env} \times R_{norm}$ is sequentially independent, and 3) the effect of each environment rule can be repaired up to normal transformations. Furthermore, an SH-System is deadlock-free, if it is normally deadlock-free, and each pair $(p,q) \in (R_{env} \cup R_{rpr}) \times R_{norm}$ is sequentially and parallel independent. For the proof of these analysis results and further self-healing and liveness properties see [EER+10].

**Tool Support**
For the automatic analysis of the static conditions ensuring the self-healing properties we use AGG, in particular to check on sequential and parallel independence of pairs of rules. AGG computes dependencies and conflicts of rules and visualizes their reasons. All properties are verified for our traffic light system.

**Related Work**
Different related approaches exist, either based on graph transformation [6,14,15,16,17,18,19] or on temporal logics and model checking [20,21,22]. In many cases, though, the state space of behavioral system models becomes too large or even infinite, and in this case model checking techniques have their limitations.

**Unsolved Problems**
A helpful extension of the formal approach would be the analysis and verification of consistency properties using the theory of graph constraints and nested application conditions in [EHL10b]. Moreover, we will investigate how far the techniques for SH-systems can be used and extended for more general self-adaptive systems.

# 7 Evaluation and Conclusion

The table in Figure 13 summarizes the problem domains and modelling features and results for our four case studies. In the last line, we state concepts which, from our point of view, are missing not only for the particular case study presented in this paper but rather in general for the respective application domain.

In the area of visual language modelling, e.g. for **Case Study (1)**, the concept of typed attributed graph transformation, which is close to meta-modelling, proved to be suitable for defining syntax and semantics of domain-specific languages. But to be useful in the context of larger systems, these principles should be integrated in tools that are used in practical applications. In our case study, a suitable user interface should hide the formal representation of abstract graph and rule syntax, and the underlying model needs to be linked to the clinic information system. Here, advanced tool support integrating graph transformation tools to existing tools used in practice is one aim for graph transformation technology transfer.

| | (1) Medical Information System | (2) Business Process Model Transformation | (3) Metabolic Pathway Analysis | (4) Self-Healing Automated Traffic Light |
|---|---|---|---|---|
| **Problem** | Adequate visualization of clinical processes | Source-to-Target model transformation | Molecular analysis of chemical reactions | System modelling with failures and recovery |
| **GraTra Model** | Visual language modelling by typed attributed GraTra | GraTra based on source-target type graph inclusion $TG_S \ddot{\mathbf{Y}} \; TG_I \hat{\mathbf{Y}} \; TG_T$ | Hypergraph transformations with simulation | GraTra with different rule sets $R_{normal}$, $R_{failure}$, $R_{repair}$ and constraints |
| **GraTra Results** | Graph constraint satisfaction after transformation | Parallel independence of amalgamated GraTra | Simulation, embedding and extension | Static analysis of self-healing properties |
| **Missing Concepts** | Advanced tool support for visual user interfaces | Semantical correctness of model trafos | Scalability of graph representation | Critical pair analysis for general conditions |

Figure 13: Comparison of Case Studies

Model transformations from domain-specific models to more machine-centric formats like **Case Study (2)** have become a necessary step towards unified and standards-based development environments. Here, important results have been achieved in recent years concerning the syntactical correctness of model transformations and their functional behaviour, i.e. termination and uniqueness. Also, for triple graph grammars, properties concerning the consistency of source and target models w.r.t. triple rules can be shown formally. An open problem for model transformations remains the semantical correctness, i.e. how can be shown in general that the behaviour of the source and the target model are equivalent (see also [Erm09]).

Often, a validation by simulation is helpful to provide new insights on behavioural system properties. **Case Study (3)** showed that a simulation by graph transformation, supported by tools, can help to find a suitable abstraction level and visualize model features (like molecule identities) which are not easily seen using standard techniques and tools. Here, the problem arises that in contrast to standard tools, a graph representation might lead to a larger memory consumption than e.g. the standard format for chemical formulae. Morover, scalability, i.e. effieciency of the rewriting in large models is a general problem. These problems have been tackled already by comparing and improving the performance of existing graph transformation engines and by experimenting with different data formats for graphs and rules and by optimizing the pattern matching process which is the bottleneck of graph transformation. Here, more future work will be necessary for further optimizations.

Many verification results for graph transformation systems are based on critical pair analysis. This kernel technique is also used in **Case Study (4)**, where we analyze conflicts and dependencies of rules to show self-healing properties. Recently, general (nested) conditions on graphs have been defined by Habel and Pennemann [HP09]. These conditions allow for a very flexible modelling of graph rules. In this context, it remains to provide a suitable theory for critical

pair analysis of rules with nested application conditions while a formal background is presented already in [EHL+10a]. .

Some of the "missing concepts" are topics of ongoing research projects[2]. We are confident that the visibility of graph transformation technology in practice will be further enhanced and that meetings between theory and practice, aided by good tool support, will be the rule rather than the exception.

## Bibliography

[AGG09] TFS-Group, TU Berlin. AGG. 2009. http://tfs.cs.tu-berlin.de/agg.

[AKL03] A. Agrawal, G. Karsai, A. Ledeczi. An End-to-End Domain-Driven Software Development Framework. In *Proc. Conf. on Object-Oriented Programming, Systems, Languages and Applications*. ACM SIGPLAN, USA, 2003.

[BEE+10] E. Biermann, H. Ehrig, C. Ermel, U. Golas, G. Taentzer. Parallel Independence of Amalgamated Graph Transformations Applied to Model Transformation. In *Graph Transformations and Model-Driven Engineering. Essays Dedicated to Manfred Nagl*. LNCS 5765. Springer, 2010. To appear.
http://tfs.cs.tu-berlin.de/publikationen/Papers10/BEE+10.pdf

[BEEH09] E. Biermann, K. Ehrig, C. Ermel, J. Hurrelmann. Generation of Simulation Views for Domain Specific Modeling Languages based on the Eclipse Modeling Framework. In Taentzer and Heimdahl (eds.), *Automated Software Engineering (ASE'09)*. Pp. 625 – 629. IEEE Press, 2009.

[BEL+10] E. Biermann, C. Ermel, L. Lambers, U. Prange, G. Taentzer. Introduction to AGG and EMF Tiger by Modeling a Conference Scheduling System. *Int. Journal on Software Tools for Technology Transfer* 12(3-4):245–261, Juli 2010.
doi:10.1007/s10009-010-0154-x
http://www.springerlink.com/content/p4n1g45627852743/

[BET08] E. Biermann, C. Ermel, G. Taentzer. Precise Semantics of EMF Model Transformations by Graph Transformation. In Czarnecki (ed.), *Proc. Int. Conf. on Model Driven Engineering Languages and Systems (MoDELS'08)*. LNCS 5301, pp. 53–67. Springer, 2008. http://tfs.cs.tu-berlin.de/publikationen/Papers08/BET08.pdf

[Béz05] J. Bézivin. On the unification power of models. *Software and System Modeling* 4(2):171–188, 2005.

[BFH87] P. Böhm, H.-R. Fonio, A. Habel. Amalgamation of graph transformations: a synchronization mechanism. *Computer and System Sciences (JCSS)* 34:377–408, 1987.

[2] See e.g. our project *Behaviour Simulation and Equivalence of Systems Modelled by Graph Transformation* (supported by the German Research Council) at http://www.tfs.tu-berlin.de/menue/forschung/#BehaviourGT.

[BNS+05] A. Balogh, A. Németh, A. Schmidt, I. Rath, D. Vágó, D. Varró, A. Pataricza. The VIATRA2 Model Transformation Framework. In *Proc. European Conference on Model Driven Architecture (ECMDA'05)*. 2005.

[BTMS99] R. Bardohl, G. Taentzer, M. Minas, A. Schürr. Application of Graph Transformation to Visual Languages. In Ehrig et al. (eds.), *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages and Tools*. World Scientific, 1999.

[EEKR99] H. Ehrig, G. Engels, H.-J. Kreowski, G. Rozenberg (eds.). *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages and Tools*. World Scientific, 1999.

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

[EER+10] H. Ehrig, C. Ermel, O. Runge, A. Bucchiarone, P. Pelliccione. Formal Analysis and Verification of Self-Healing Systems. In *Proc. Int. Conf. on Fundamental Aspects of Software Engineering (FASE'10)*. LNCS 6013, pp. 139–153. Springer, 2010. http://www.springerlink.com/content/hv51032524v38321/

[EHL06] K. Ehrig, R. Heckel, G. Lajios. Molecular Analysis of Metabolic Pathway with Graph Transformation. In *Proc. Int. Conf. on Graph Transformation (ICGT'06)*. LNCS 4178, pp. 107–121. Springer, 2006.

[EHL+10a] H. Ehrig, A. Habel, L. Lambers, F. Orejas, U. Golas. Local Confluence for Rules with Nested Application Conditions. In *Proc. Int. Conf. on Graph Transformation (ICGT'10)*. 2010. To appear. http://tfs.cs.tu-berlin.de/publikationen/Papers10/EHL+10.pdf

[EHL10b] H. Ehrig, A. Habel, L. Lambers. Parallelism and Concurrency Theorems for Rules with Nested Application Conditions. *Electr. Comm. of the EASST* 26, 2010. http://journal.ub.tu-berlin.de/index.php/eceasst/issue/view/36

[EKMR99] H. Ehrig, H.-J. Kreowski, U. Montanari, G. Rozenberg (eds.). *Handbook of Graph Grammars and Computing by Graph Transformation. Vol 3: Concurrency, Parallelism and Distribution*. World Scientific, 1999.

[EMF09] Eclipse Consortium. Eclipse Modeling Framework Technology. 2009. http://www.eclipse.org/modeling/emft.

[Eng00] G. Engels. Graph Changes are Everywhere: The Role of Graph Transformations in Software Engineering. In *Proc. Joint APPLIGRAPH and GETGRATS Workshop on Graph Transformation Systems*. pp. 12-13. TU Berlin, 2000.

[Erm09] C. Ermel. Visual Modelling and Analysis of Model Transformations based on Graph Transformation. *Bulletin of the EATCS* 99:135 – 152, 2009.

[GEH10] U. Golas, H. Ehrig, A. Habel. Multi-Amalgamation in Adhesive Categories. In *Proc. Int. Conf. on Graph Transformation (ICGT'10)*. 2010. To appear. http://tfs.cs.tu-berlin.de/publikationen/Papers10/GEH10.pdf

[GMF07] Eclipse Consortium. Eclipse Graphical Modeling Framework (GMF). 2007. http://www.eclipse.org/gmf.

[HP09] A. Habel, K.-H. Pennemann. Correctness of high-level transformation systems relative to nested conditions. *Mathematical Structures in Comp. Science* 19:1–52, 2009.

[IBM03] IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems. Business Process Execution Language for Web Services version 1.1. May 2003. http://www.ibm.com/developerworks/library/ws-bpel/.

[LETE04] J. de Lara, C. Ermel, G. Taentzer, K. Ehrig. Parallel Graph Transformation for Model Simulation applied to Timed Transition Petri Nets. *ENTCS* 109:17–29, 2004. http://tfs.cs.tu-berlin.de/publikationen/Papers04/LETE04.pdf

[Löw93] M. Löwe. Algebraic Approach to Single-Pushout Graph Transformation. *TCS* 109:181–224, 1993.

[LVA04] J. de Lara, H. Vangheluwe, M. Alfonseca. Meta-Modelling and Graph Grammars for Multi-Paradigm Modelling in AToM$^3$. *Software and System Modeling: Special Section on Graph Transformations and Visual Modeling Techniques* 3(3):194–209, 2004.

[Min07] M. Minas. DiaGen / DiaMeta – The Diagram Editor Generator. 2007. http://www.unibw.de/inf2/DiaGen/.

[Mos96] G. Moss (ed.). *IUPAC Basic Terminology of Stereochemistry*. Volume 68(12). Pure & Applied Chemistry, 1996.

[MVVK05] T. Mens, P. Van Gorp, D. Varrò, G. Karsai. Applying a Model Transformation Taxonomy to Graph Transformation Technology . In *Proc. Int. Workshop on Graph and Model Transformation (GraMoT'05)*. ENTCS 152, pp. 143–159. Elsevier Science, 2005.

[OMG07] Object Management Group. Unified Modeling Language: Superstructure – Version 2.1.1. 2007. http://www.omg.org/technology/documents/formal/uml.htm.

[Ope09] OpenEmbeDD: Model Driven Engineering open-source platform for Real-Time & Embedded systems. 2009. http://openembedd.org.

[Roz97] G. Rozenberg. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, 1997.

[RV05] F. Rosselló, G. Valiente. Graph Transformation in Molecular Biology. In *Formal Methods in Software and System Modeling, LNCS 3393*, pp. 116–133. Springer, 2005.

[RV10] A. Rensink, P. Van Gorp (eds.). *International Journal on Software Tools for Technology Transfer (STTT), Special Section on Graph Transformation Tool Contest 2008*. Volume 12(3-4). Springer, 2010.
http://www.springerlink.com/content/p4n1g45627852743/

[SAL+03] J. Sprinkle, A. Agrawal, T. Levendovszky, F. Shi, G. Karsai. Domain model translation using graph transformations. In *Int. Conf. on Engineering of Computer-Based Systems*. Pp. 159–168. 2003.

[Sch94] A. Schürr. Specification of Graph Translators with Triple Graph Grammars. In *WG94 20th Int. Workshop on Graph-Theoretic Concepts in Computer Science*. LNCS 903, pp. 151–163. Springer, 1994.

[Tae04] G. Taentzer. AGG: A Graph Transformation Environment for Modeling and Validation of Software. In *Application of Graph Transformations with Industrial Relevance (AGTIVE'03)*. LNCS 3062, pp. 446 – 456. Springer, 2004.

[Tae06] G. Taentzer. Characterizing Tools for Visual Modeling Techniques. In Ehrig et al. (eds.), *Lecture Notes of SegraVis Advanced School on Visual Modelling Techniques*. Univ. of Leicester, 2006.
http://tfs.cs.tu-berlin.de/publikationen/Papers06/Tae06a.pdf

[TEG+05] G. Taentzer, K. Ehrig, E. Guerra, J. de Lara, L. Lengyel, T. Levendovsky, U. Prange, D. Varro, S. Varro-Gyapay. Model Transformation by Graph Transformation: A Comparative Study. In *Proc. Workshop Model Transformation in Practice*. 2005.
http://tfs.cs.tu-berlin.de/publikationen/Papers05/TEG+05.pdf

[TR03] J. Tolvanen, M. Rossi. MetaEdit+: Defining and Using Domain-Specific Modeling Languages and Code Generators. In *Proc. Conf. on Object-oriented programming, systems, languages, and applications (OOPSLA '03)*. Pp. 92–93. ACM Press, 2003.

[VSV05] G. Varró, A. Schürr, D. Varró. Benchmarking for Graph Transformation. In *Proc. IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 05)*. Pp. 79–88. IEEE Press, 2005.

[Whi04] S. White. Business Process Modeling Notation (BPMN) Version 1.0. BPMI.org, 2004.

[ZPV95] G. Zubay, W. Parson, D. Vance. *Principles of Biochemisty*. Volume 2. McGraw-Hill College, 1995.