Frank Hermann

Analysis and Optimization of Visual Enterprise Models

Based on Graph and Model Transformation

Universitätsverlag der TU Berlin

ISBN 978-3-7983-2321-6 (Druckausgabe)
ISBN 978-3-7983-2322-3 (Online-Version)

Berlin 2011

∞ Gedruckt auf säurefreiem alterungsbeständigem Papier

Druck/ Printing: Endformat, Ges. für gute Druckerzeugnisse mbH
Köpenicker Str. 187-188, 10997 Berlin

Verlag/ Publisher: Universitätsverlag der TU Berlin
Universitätsbibliothek
Fasanenstr. 88 (im VOLKSWAGEN-Haus), D-10623 Berlin
Tel.: (030)314-76131; Fax.: (030)314-76133
E-Mail: publikationen@ub.tu-berlin.de
http://www.univerlag.tu-berlin.de

Analysis and Optimization of Visual Enterprise Models Based on Graph and Model Transformation

vorgelegt von
Diplom-Informatiker
Frank Hermann
aus Luckenwalde

Von der Fakultät IV - Informatik und Elektrotechnik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften
Dr.-Ing.

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Dr. rer. nat. Bernd Mahr
Berichter:	Prof. Dr. rer. nat. Hartmut Ehrig
Berichterin:	Prof. Dr. rer. nat. Barbara König
Berichter:	Prof. Dr. rer. nat. Thomas Engel

Tag der wissenschaftlichen Aussprache: 10. 02. 2011

Berlin 2011
D 83

Abstract

Security, risk and compliance are increasingly important issues in enterprise modelling. An analysis of real world scenarios for banking has shown that today's "best practices" based on informal techniques are insufficient and should be replaced by adequate formal approaches. For this purpose, visual enterprise models based on algebraic graph and model transformation are proposed in this thesis, where the following two typical problem areas are considered:

1. Behaviour Analysis and Optimization of Visual Enterprise Process Models
2. Conformance Analysis of Enterprise Process and Service Models

The first problem is solved by new concepts and results concerning behaviour analysis of visual languages, especially analysis of permutation equivalence of graph transformations based on Petri nets. In order to tackle the second problem area, model transformations based on triple graph grammars are introduced with new results concerning analysis and optimization enabling automated support for sound integration and conformance checks between different models.

The new results in the area of graph and model transformation are developed in the general framework of $\mathcal{M}$-adhesive transformation systems. They can be instantiated to a large variety of high-level replacement systems and applied not only to enterprise modelling, but also to several other application domains. Last but not least, the new tool AGT-M provides automated tool support for simulation, analysis and optimization in various case studies.

Zusammenfassung

Sicherheit, Risikomanagement sowie Regelkonformität sind von wachsender Bedeutung im Bereich der Unternehmens-Modellierung. In einer Analyse von realen Szenarien im Bankenbereich wurde deutlich, dass heutige "'Best Practices"', die auf informellen Techniken basieren, nicht ausreichen und durch adäquate formale Techniken ersetzt werden sollten. Zu diesem Zweck schlagen wir die visuelle Modellierung von Unternehmen basierend auf algebraischer Graph- und Modelltransformation vor, wobei die beiden folgenden typischen Problemfelder betrachtet werden:

1. Verhaltensanalyse und Optimierung von visuellen betriebswirtschaftlichen Prozess-Modellen
2. Konformitätsanalyse zwischen betriebswirtschaftlichen Prozess- und Organisations-Modellen

Das erste Problem wird mit Hilfe neuer Konzepte und Resultate bezüglich der Verhaltensanalyse von visuellen sprachen gelöst, insbesondere mit Hilfe der Analyse von Permutations-Äquivalenz von Graphtransformationen basierend auf Petrinetzen. Zur Behandlung des zweiten Problembereichs werden Modelltransformationen basierend auf Triple-Graphgrammatiken inklusive neuer Resultate für die Analyse und Optimierung vorgestellt, was eine automatisierte Unterstützung einer konsistenten Integration sowie Konformitätsüberprüfung zwischen den Modellen ermöglicht.

Die neuen Resultate im Bereich der Graph- und Modelltransformation werden in dem allgemeinen Rahmenwerk der $\mathcal{M}$-adhesiven Transformationssysteme entwickelt. Sie können somit für zahlreiche High-Level-Ersetzungssysteme instanziiert werden und demzufolge nicht nur im Bereich der Unternehmensmodellierung sondern auch in diversen anderen Anwendungsdomänen eingesetzt werden. Zudem bietet die neue Software AGT-M eine automatisierte Werkzeugunterstützung für die Simulation, Analyse und Optimierung in verschiedenen Fallstudien.

Acknowledgements

The research for this thesis has been carried out at the institute for Theoretical Computer Science and Software Engineering at the Berlin Institute of Technology (Technische Universität Berlin) in Germany and partly at the university of Pisa (Università di Pisa) in Italy as well as at the technical university in Barcelona (UPC - Universitat Politècnica de Catalunya) in Spain.

For his highly valuable, enthusiastic, and motivating support during the supervision of my research, I would like to thank *Hartmut Ehrig*. He introduced me to the field of graph and model transformation with its extensive formal foundations and powerful capabilities for many application domains. During my studies, I highly enjoyed the fruitful discussions, interesting projects and joint contributions to the international scientific community.

For all their valuable support and the fruitful cooperation in joint work for several interesting conferences and workshops, I would also like to thank my co-supervisors *Barbara König* and *Thomas Engel*.

For the interesting as well as enlightening discussions and for the highly successful joint work, I would like to thank *Andrea Corradini* and *Fernando Orejas*, who additionally supported my international stays at University of Pisa and UPC Barcelona.

Finally, I would like to sincerely thank all my colleagues for the inspiring time, the fruitful discussions and the highly successful cooperation.

Contents

Chapter 1

Introduction

Visual modelling is a fundamental and important concept in a large variety of research areas, such as architecture, biology, and computer science. The visual models are used in these fields to provide a suitable abstraction of the modelled objects. Based on these abstractions, the involved persons can obtain a comprehensive overview of the modelled systems and artefacts, which additionally requires a common understanding of the visual notation. Depending on the underlying modelling concepts, the analysis of models with respect to certain specified requirements and criteria can be supported by automated and verified techniques. However, visual modelling techniques often do not provide these capabilities. The main reason is that their human-centric notation is usually not equipped with an underlying formal notation, which would be adequate for currently available analysis techniques.

From a general point of view, modelling languages can be formal (like the process algebra CSP [Hoa85]), informal (like specifications in natural language), or semi-formal (like UML [OMG10]), which means that their syntax or semantics is defined only partially by formal methods, such that different or even incompatible interpretations are possible. Besides general-purpose modelling languages like UML [OMG10], domain specific modelling languages (DSMLs) are increasingly used in model driven software development, because they provide more specific concepts and notations. Therefore, DSMLs build the basis for an improved understanding and more powerful code generators [TK09]. Several meta tools are available, which provide a DSML specification environment for rapid DSML tool generation, like MetaEdit+ [Tol07], DiaGen [Min07], AToM 3 [dV07], GME [LBM+01], Marama [GHHN08], Microsoft Visual DSL Tools [Mic10] and EMF Tiger [Tig10]. This means that organization specific DSMLs can be developed and used without additional efforts for developing the modelling environments.

Domain specific modelling languages for visual enterprise models usually have a strong focus on human-centric notation, because they are used for modelling structures and activities of enterprises including the human actors. While such DSMLs have been successfully

applied for improving the design and documentation of business objects and operations based on their intuitive notations, the developed models are used only partially for analysis and administration of business operations. However, improved analysis and administration capabilities do also belong to the central aims of enterprise modelling [FG98, SAB98].

Today, security, risk and compliance are increasingly important issues in enterprise modelling. An analysis of real world scenarios for banking in an ongoing project with Credit Suisse and University of Luxembourg [BBE+07, BHE09a, BHE09b, BHEE10] has shown that current "best practices" based on informal techniques are insufficient and should be replaced by more adequate formal approaches. A very common practice today is to provide guidelines of best practices and to judge about security, risk and compliance requirements by evaluating check lists. This pragmatic approach, however, does not ensure that the requirements are respected.

Suitable formal techniques, however, can build the basis for fully automated verification. Therefore, domain specific modelling languages for organizational aspects need to be formal in order to enable the verification of requirements. Moreover, they need to be expressive enough in order to describe the static and dynamic aspects of an enterprise [PS07]. These requirements are especially important in the financial sector, which is faced with an increasing complexity of regulations. Security requirements in this context concern the whole information flow based on different classifications of information from confidential to public, but also based on specific dependencies between related data objects. Risk in general, concerns all kinds of possible events that may lead to a loss of profit or even of business activity. Finally, compliance requires that an enterprise or a specific department adheres to certain internal guidelines, external regulations and laws.

Because of the extensive formal foundation available for the framework of algebraic graph transformation [Roz97, EEPT06], we propose to use graph transformation techniques for the analysis of enterprise models based on the abstract syntax graphs of the visual models. The universal visual and formal technique of graph transformation has been already applied for modelling as well as analysing distributed and concurrent systems, for the definition of visual languages and for specifying, analysing as well as performing model transformations. In addition, the theory of algebraic graph transformation provides several important and general results for analysing transformation systems concerning the combination of transformation steps (Local Church-Rosser Thm., Concurrency Thm., Embedding and Extension Thm.) and several results concerning confluence analysis (Termination, Critical Pair Lemma and Completeness of Critical Pairs) [EEPT06, Lam09]. Furthermore, properties of visual models can be formalized as graph constraints [EEPT06, HP09], which are shown to be equivalent to first order logic on graphs [Ren04a], and they can be verified by automated techniques [AGG10, Pen09].

In order to provide powerful techniques for the analysis of enterprise models, we can apply on the one hand existing results. On the other hand, we are able to provide addi-

tional new techniques leading to new results and applications. Due to the general character of graph transformation, these new techniques can be used for further domains of visual modelling as well. Based on the extended results, we will perform formally well-founded analysis techniques on visual human-centric models and derive formally verified results. In particular, we propose to formalize *security* requirements as graph constraints – where possible – and check them using formal and automated techniques.

In the area of *risk* management we will focus on the availability of actors and resources of an enterprise. In the case that some of them are not available, running business processes can fail and therefore, the enterprise has to provide an effective *business continuity management* to ensure continuity of critical business processes that are vital for the survival of the enterprise. However, the combination of different possible failures leads to a high complexity and therefore, the manual modelling of complete business continuity processes is only possible for a minor subset of the possible combinations. Instead, we can show that the modelling of business processes including business continuity aspects can be improved and supported by automated techniques.

In order to support the analysis of *compliance* and to improve the overall modelling process, we propose to apply model transformations based on triple graph grammars (TGGs) [Sch94, SK08]. While the notion of compliance covers many different aspects, we focus firstly on checking those constraints that can be formalized as graph constraints and secondly on analysing and checking *conformance* between different heterogeneous enterprise models. In order to ensure efficient and verified conformance checks we have to extend the available TGG results and develop adequate execution and analysis techniques.

Aims of the Thesis

In order to support the capabilities of visual enterprise modelling, we develop, on the one hand, improved modelling concepts and extensions and, on the other hand, we provide suitable formal techniques and results that can be efficiently applied for analysing the visual enterprise models. The first main challenge in this context is to provide an adequate formal and efficient technique for the analysis and optimization of visual business process models including the aspects of business continuity management. The second main challenge is the analysis of conformance between business process and business service models, in order to formally verify that the specified restrictions and permissions within the business service models are respected by the business process models.

According to the scope of the thesis in Fig. 1.1, we consider visual human-centric business process models and business service models, but the techniques are developed on a general basis allowing for applications in further domains. The specific DSML for business processes in our case study is an extended version of event driven process chains (EPCs)

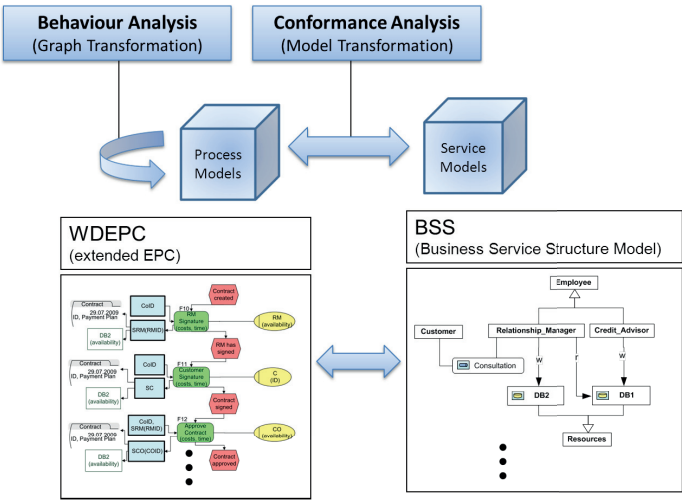


Figure 1.1: Scope of the Thesis

called workflow engine and data oriented EPCs (WDEPCs). This extension provides additional information, which is relevant for a sound analysis of standard and continuity processes and for supporting recommender services during the execution of business processes. The models in the business service domain specify business objects, hierarchies or matrices, communication channels and access rights. However, these four aspects can also be developed in separate models, such that they are either integrated when needed or analysed separately.

The main aims of this thesis in the area of enterprise modelling and in the general context of visual modelling are specified by the following list of tasks:

1. Provide formal and efficient techniques for analysing the behaviour of graph transformation systems with respect to possible equivalent reorderings of transformation sequences.

2. Develop an efficient modelling framework for business process models based on a human-centric visual notation and an underlying formal abstract syntax for formal analysis and verification using the techniques of 1.

3. Extend the results for model transformations based on TGGs in order to ensure syntactically correct, complete and efficient executions. Provide further important analysis techniques.

4. Develop concepts for checking conformance between heterogeneous models. Provide results that ensure efficient and verified checks using the results in 3.

5. Provide the formal results in the general framework of high level replacement systems ($\mathcal{M}$-adhesive transformation systems), such that they can be instantiated in several important domains.

6. Apply the results to realistic case studies and provide prototypical tool support.

Main Results

According to the following main results of the thesis the specified aims above are realized, such that the developed techniques can be applied for improved enterprise modelling and analysis as well as for improving visual modelling in other domains, like model driven software development.

Behaviour Analysis of Transformation Systems The behaviour of transformation systems is analysed based on its interleaving semantics, which specifies the equivalent reorderings of a given transformation sequence. In order to include transformation rules which contain negative application conditions (NACs) – a widely used control structure – we introduce the notion of permutation equivalence and provide a generalised construction of processes (Def. 3.3.25), which consist of a subobject transformation system (STS) together with an embedding into the original transformation system. The generation of a process is ensured to be performed in polynomial time and the analysis based on STSs is shown to be correct and complete (Thm. 3.4.12). Moreover, we present the generation of the dependency net of a process and show that the analysis based on the dependency net is correct and complete (Thm. 3.5.3) as well as efficient according to the provided benchmark.

Business Process Modelling A new concept of continuity snippets for specifying alternative process fragments, which handle specific failure cases, improves the efficiency and quality of business process modelling and business continuity management. The presented analysis techniques for business processes models are used to verify the formalized functional requirements (given by sets of required data elements and events) and non-functional requirements (given by security graph constraints). The correctness is based on the formal results for behaviour analysis of transformation systems above. We further show that the generation of process runs and continuity process runs ensures the formalized functional and non-functional requirements (Thms. 4.2.7, 4.2.10) Furthermore, we illustrate the application of the analysis and generation techniques for the case study based on the provided prototypical tool support.

Model Transformation We provide a formal and efficient approach to model transformation based on triple graph grammars (TGGs) and show that most of the general challenges for model transformations concerning functional and non-functional criteria (see Sec. 5.1) can be mastered by the approach. The execution ensures syntactical correctness and completeness (Thm. 5.2.2, Cor. 5.2.6) and we provide static conditions for guaranteeing termination (Thms. 5.2.4, 5.2.14), (strong) functional behaviour (Thms. 5.2.27, 5.2.31), and (complete) information preservation (Thms. 5.2.34, 5.2.37). Moreover, we present optimization techniques for model transformations concerning the conservative conflict resolution (Thm. 5.3.7) and the improvement of efficiency by reducing and – if possible – eliminating backtracking (Thms. 5.3.10, 5.2.27).

Conformance Analysis In order to analyse conformance between models of heterogeneous DSMLs we provide different notions of conformance (conformance, forward conformance, and restricted forward conformance) as well as efficient techniques for checking conformance (Thms. 6.3.3, 6.4.1) based on the results for model transformation. Furthermore, we show the applicability of the techniques by verifying that the business process model of our case study conforms to the given business service structure model via the notion of restricted forward conformance. This ensures that the specified permissions for communication and access to resources are respected for any execution of the process model and its continuity alternatives.

General Framework The new results in the area of graph and model transformation are developed in the general framework of $\mathcal{M}$-adhesive transformation systems. They can be instantiated to a large variety of high-level replacement systems and applied not only to enterprise modelling, but also to several other application domains like model driven software development and, moreover, for the behaviour analysis of mobile ad hoc networks modelled by reconfigurable Petri nets.

Case Studies and Tool Support The provided techniques and results are applied in several case studies: behaviour analysis of a workflow management system, analysis and optimization of a loan granting process, model transformation from class diagrams to relational data base models, conformance analysis of business process and business service structure models. Moreover, the techniques are executed based on the provided tool AGT-M together with the analysis engine of the tool AGG.

Overview of the Chapters

Chapter 2 (Visual Enterprise Modelling) presents the main challenges in enterprise modelling and discusses the benefits of using visual enterprise modelling techniques based on formal methods. In particular, we discuss main problems for large multi-national enterprises in the financial sector based on an ongoing research project with Credit Suisse and University of Luxembourg. As a key challenge in this context, human-centric visual models have to adhere to external regulations while they are distributed over departments and shall be easy to maintain. In order to improve enterprise modelling in this domain, the subsequent chapters present new and formally well founded techniques, for which we show that the application of these techniques is adequate and efficient in this and in other domains like model driven software development.

Chapter 3 (Behaviour Analysis of Visual Languages Based on Graph Transformation) introduces an extended formal framework for the analysis of behavioural models based on their underlying abstract syntax graphs and additional operational semantic rules. The formal results concerning the analysis of interleaving semantics are shown in the general framework of $\mathcal{M}$-adhesive transformation systems and are instantiated to typed attributed graph transformation systems. In particular, we introduce the new notion of permutation equivalence [Her09a] as a generalisation of the standard notion of switch equivalence in order to analyse more complex transformation systems. The analysis of interleaving semantics is based on a generalised notion of processes for $\mathcal{M}$-adhesive transformation systems using the concept of subobject transformation systems [CHS08]. Furthermore, we show that the analysis can be performed efficiently based on the generation of a compact process model given by a P/T Petri net – called dependency net [HCEK10a].

Chapter 4 (Behaviour Analysis and Optimization of Visual Enterprise Process Models) presents improved modelling concepts in the area of business process modelling in order to improve business process analysis and to support business continuity management. Instead of modelling complete business continuity processes, we show that it is sufficient and more efficient to model only fragments of continuity processes which are specific to the different possible failures [BHG11]. Based on these fragments, called continuity snippets, we provide an automated technique for the generation of validated complete business process executions for the different possible combinations of failures. The case study shows how a loan granting process is analysed with respect to functional and non-functional requirements based on a given human-centric process model in the domain specific language of extended event driven process chains. For this purpose, we provide a formal operational semantics given by a subobject transformation system derived from the abstract syntax graphs of the visual models and we apply the analysis techniques

of Ch. 3. Exemplarily, we verify the formalized security requirement “four-eye principle” and generate more than 100 valid process runs and more than 200 additional continuity process runs that satisfy the given functional and non-functional requirements.

Chapter 5 (Model Transformation Based on Triple Graph Grammars) discusses main challenges of model transformations and provides a powerful approach to bidirectional model transformations based on triple graph grammars. Model transformations are a key concept in model driven development and we show that the achieved results in this thesis allow us to master most of the discussed challenges for model transformations. In particular, the approach ensures syntactical correctness, completeness and we provide efficient static criteria for ensuring termination [EHS09a]. Moreover, several analysis techniques are presented for checking and improving important properties of model transformations. This includes (strong) functional behaviour [HEGO10c], (complete) information preservation [EEE+07] and efficiency [HEGO10c]. The case study on the well-known model transformation from class diagrams to relational data base models shows that the techniques can be efficiently applied and used to improve the system of model transformation rules, such that the discussed properties can be verified.

Chapter 6 (Conformance Analysis of Enterprise Process and Service Models) addresses the problem of checking and ensuring conformance of enterprise models in different heterogeneous domain specific modelling languages. Conformance analysis is especially important between business process and business service structure models, which specify certain restrictions and permissions concerning internal communication and access to resources. Based on the abstract syntax of the visual models we show how the developed techniques and results in Ch. 5 for model transformations are efficiently used to check conformance for the case study. The presented automated and efficient conformance analysis techniques are suitable for a decentralised modelling process, which is important for large enterprises, where support of decentralisation is a highly relevant requirement.

Chapter 7 (Prototypical Tool Support: AGT-M) provides an overview of the developed tool AGT-M and the used features of the tool AGG and compares their capabilities with other available tools.

Chapter 8 (Related Work) discusses and compares related approaches and techniques in the different areas concerning Chapter 3 to 6, while related work for enterprise modelling in general and tool support are discussed in Chs. 2 and 7 already.

Chapter 9 (Conclusion and Future Work) summarizes the main achievements of the thesis in the area of visual enterprise modelling and software driven development and presents possible further application domains and extensions for future work.

Chapter 2

Visual Enterprise Modelling

The aim of enterprise modelling is to support and improve the design, documentation, analysis and administration of business objects and operations based on adequate modelling techniques [FG98, SAB98]. For this purpose, domain specific enterprise models shall provide the basis for communication between people with different professional background [Fra02]. This chapter presents main aspects and problems of enterprise modelling and specifies the scope of the developed techniques and applications of this thesis within the wide area of enterprise modelling.

Enterprise models provide representations of the structures, processes, resources, involved actors, executed functions, goals, and constraints relevant for the modelled enterprise. For this reason, enterprise modelling has to provide an agile modelling process, which is integrated across the different business functions [FG98]. An agile modelling process additionally reduces the required time frames for adapting the models according to change requests, which can occur quite frequently during the lifetime of an enterprise. Moreover, adequate modelling techniques should support the propagation of changes from one domain to other domains. This way, the knowledge and expertise of enterprise modellers can be focussed on their main domain, which is an important requirement for decentralised and distributed models occurring especially in large multi-national enterprises.

In order to master the high complexity of an enterprise in its whole, visual modelling techniques have been successfully applied. They provide intuitive notations and high abstraction capabilities. Clearly, visual models cannot replace all textual models in all domains. But still, where suitable, they often show high benefits. Furthermore, enterprise models are usually not only used for the design and documentation of enterprises, but also for the analysis and management of operations. In particular, process analysis concerns, e.g., the question whether certain business processes can be performed in a different but more suitable way, such that some goals can be achieved in an optimized way.

The integration of different enterprise models requires, on the one hand, the application of techniques that ensure certain quality and consistency requirements and, on the other hand, the application of techniques for setting up and maintaining a common understanding of the enterprise by the modellers. While this thesis provides several suitable techniques for the first requirement, the common understanding is supposed to be set up and maintained based on sophisticated techniques in the area of ontology engineering [FG98].

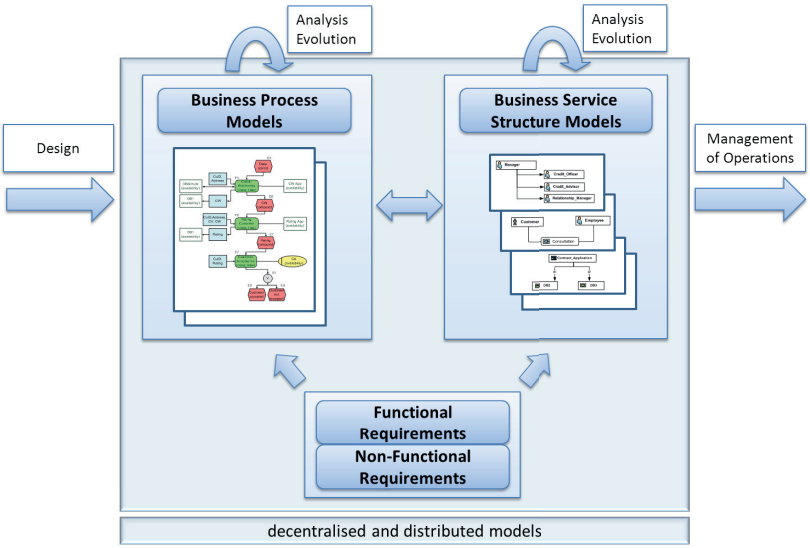


Figure 2.1: Scope of enterprise modelling in this thesis

This thesis is focussed on aspects in enterprise modelling especially relevant for enterprises in the financial area, such that the scope covers business process and business service structure models as shown by the overview in Fig. 2.1. Accordingly, the case studies in Chapters 4 and 6 are placed within the financial area and they were developed in a fruitful cooperation with the University of Luxembourg and Credit Suisse. As the figure shows, the main goal is to provide suitable techniques for the design (incoming left arrow) in order to effectively support the management of business operations (outgoing right arrow).

Business process models (top left of Fig. 2.1), like event driven process chains (EPCs) [IDS10, Men07] or models in business process modelling notation (BPMN models, [OMG09]), are especially important for enterprise modelling in the financial domain, where business processes need to be evaluated and have to be ensured to respect a wide range of non-functional requirements, like for instance legislative restrictions and obliga-

tions. Business service structure models (top right of Fig. 2.1) are used for specifying the organizational structure and roles of actors within an enterprise. Both model types evolve over time and need to be analysed and maintained in order to ensure and improve correct business activities. This includes the validation of given functional and non-functional requirements, which have to be respected because of internal business goals or because of external restrictions.

Clearly, there are several further perspectives and aspects within an enterprise which are also relevant for enterprise modelling, but they go beyond the scope of this thesis. However, the general techniques provided in this thesis show good potential that they can also improve the modelling process in further domains as discussed in [BH10]. One comprehensive framework is presented in [Fra02] based on the concept of multi-perspective enterprise modelling (MEMO). The MEMO method provides semi-formal domain specific modelling languages that are focussed on the concrete modelling domains of enterprises. Furthermore, the ISO standard 19439:2006 [ISO06] specifies a general framework for enterprise modelling and enterprise integration based on the GERAM framework (generalised enterprise reference architecture and methodology, [BN96]), where several aspects and dimensions are distinguished in a partly similar way as in the MEMO method.

Compared with enterprise modelling based on general modelling languages like UML, as presented in [Mar00], domain specific modelling techniques provide specific modelling concepts and notations which are appropriate for the modelling tasks instead of providing generic ones from which the specific ones have to be manually constructed and derived. For this reason, the unified enterprise modelling language [Ver02, ABV07] does not aim to provide one universal enterprise modelling language, but rather provides techniques for integrating models of existing domain specific languages based on merging their ontologies. This line is also followed in this thesis, where we additionally provide concepts for the concrete structural correspondences between the domain models in Chapters 5 and 6 by defining model transformations based on the underlying abstract syntax graphs of the models.

The first component of the enterprise modelling scope in Fig. 2.1 concerning business process modelling includes the modelling of standard and active business processes as well as optimized future business processes and business continuity processes for handling failures and disasters. This combination causes high dependencies between the different types of processes, because changes in the standard process can have high impact on the associated optimized and continuity process models. For this reason, the modelling of business processes can be significantly improved by an automated support based on formally well founded techniques. This way, the complexity and possible inconsistencies can be reduced. Moreover, the business process models can be used to actually control and run the real business processes based on some type of workflow engines that allow for process model input and maintanence.

Business service structure models in the second component of the enterprise modelling scope in Fig. 2.1 are used for the specification of the organisational structure, communication structures as well as roles and access rights to resources. The organisational structure includes the hierarchy or matrix relations between the employees and managers who are distributed over the different departments. Therefore, business service structure models make explicit the existing hierarchies, the used permissions as well as the available structures and resources. These information are used for checking conformance of business process models with respect to given business service structure models. For example, actors within a business process model who access certain data from some data bases must own a role to which the appropriate access rights to these resources are assigned.

The lower component of Fig. 2.1 consists of additionally specified functional and non-functional requirements. Functional requirements for process models define the minimal effects of business processes, which have to be provided and functional requirements for business service structure models specify the types of roles, communication and data manipulation which are required in order to perform the relevant business operations. Therefore, the modelling process has to support the analysis of the models with respect to the given functional requirements. Moreover, there are non-functional requirements, which usually restrict possible business operations to satisfy certain security side constraints and to meet some goals concerning efficiency, quality, and profit potential.

Large enterprises become more and more complex and enterprise modelling has to cope with a variety of large structures. The complexity does not only grow with the amount of employees, but especially with the amount of products, product features, composed IT systems, heterogeneous software components and information systems, changing requirements, and, in particular, decreasingly short development cycles in combination with a long running customer support. In order to control these complex structures, there is a special need for scalable abstraction techniques, which can be supported by visual modelling techniques, where models are synchronized and reflect the current state, structure and possible future activities of the enterprise.

As motivated before, the modelling process should be agile in order to cope with possibly frequent external change requests. For this reason, changes of requirements should not cause massive manual modifications of a large amount of models. Low manual change efforts also reduce the amount of inconsistencies causes by manually maintained models with duplicated information parts. Moreover, models need to be easy to grasp in order to improve the efficiency and quality of modelling. Visual modelling techniques in combination with suitable abstraction techniques can build the basis for solving these problems and challenges by providing concise and compact models. In combination with formal abstract syntax definitions, visual modelling techniques can additionally enable verified automated analyses, which can support error detection and thus, quality assurance of the models.

Based on the abstract syntax of visual domain specific languages, different types of visual models can be aligned, like for instance human-centric and machine-centric models. In [BBE+07], we presented a formal framework for language families and their integrated views concerning well-founded domain languages for secure coarse-grained IT system modelling. This approach was extended in [BHE09b, BH10], where we presented how security requirements for machine-centric IT and business models in the financial domain can be formalized by graph constraints (see Ch. 4) and analysed by graph transformation tools, like e.g. AGG [AGG10]. Furthermore, we applied model transformation techniques based on triple graph grammars as presented in Sec. 5.1.1 for ensuring consistency between these models and checking consistency based on model integration.

In the following, we summarize the main requirements for enterprise modelling and specific problems within the scope of this thesis and outline how the requirements are met and the problems are solved by the techniques and results in Chapters 3 to 7.

General Requirements and Provided Results for Enterprise Modelling

As discussed in detail before, the benefits of enterprise modelling can be substantially improved by providing techniques that satisfy the following requirements. The modelling process should be agile and decentralised, changes should not cause massive manual modifications, the applied techniques should enforce a low rate of occurring inconsistencies and duplicated information parts. Moreover, the techniques shall be applicable to distributed models. and the models itself shall be developed in a visual, human-centric and domain-specific notation. The modelling framework shall build the basis for supporting the management of operations and for analysing and ensuring functional and non-functional requirements.

In order to satisfy these requirements in a suitable way we apply formal techniques based on graph and model transformation (Chapters 3, 5), which provide powerful and efficient analysis techniques. Their formal foundation ensures correct analysis results and the automated tool support provides efficient checks. The techniques are applied to enterprise models in Chapters 4 and 6 based on the abstract syntax graph of the human-centric visual models of common domain specific modelling languages. Moreover, the model transformation techniques provide the basis for and agile and decentralised modelling process including distribution of models and efficient change propagation. The new concepts in Ch. 4 based on continuity snippets reduce the manual modelling efforts and possible inconsistencies as well as duplicates.

Specific Problems and Solutions in Business Process Modelling

Business process models need to be analysed in order to derive equivalent and valid executions of the process, which can be ranked according to some optimization criteria regarding costs and time. The process models additionally have to satisfy functional requirements (given by minimal required effects) and non-functional requirements (concerning e.g. security, efficiency or quality). In order to ensure the effectiveness of a corresponding business continuity management, the continuity process have to modelled as well, which leads to high dependencies between standard process models and the continuity process models, such that changes can have high impact. Moreover, scalability is important, because the amount of available processes can be high.

Based on the formal techniques and results on behaviour analysis in Ch. 3 we show in Ch. 4 how business process models are analysed by automated and implemented techniques that generate complete sets of equivalent process executions, which are additionally verified to ensure the formalized functional and non-functional requirements (formalized as visual graph constraints). Moreover, we show how complete continuity processes for different combinations of failures can be generated out of the standard process model together with additional continuity snippets that concern only specific failures. This way, dependencies between the models and change impacts are reduced substantially. Moreover, in combination with the benchmark in Ch. 3 the overall approach shows good scalability.

Specific Problems and Solutions in Conformance Analysis

The different domain specific modelling languages in enterprise modelling cause additional efforts for the sound integration of the existing enterprise models. Besides business process models we consider business service structure models (BSS models), which also have to adhere and ensure functional requirements (availability of certain roles, communication structure and system resources) as well as non-functional requirements (security constraints).

In Ch. 2, we present techniques or checking conformance between different heterogeneous models based on the formal techniques for model transformation in Ch. 5, for which we provide efficient tool support in Ch. 7. This way, business process models are analysed whether they conform to the given BSS models that specify permissions regarding the use of communication channels and resources by the involved actors. Moreover, we show that the model transformation techniques can be applied to generate models in related domains, which automatically ensure conformance and can be used for further refinement. Furthermore, the presented type of BSS models in Ch. 6 explicitly specifies existing hierarchies, available structures, resources and access rights. Therefore, the functional and non-functional requirements can be analysed by formalizing the requirements as visual graph constraints on the structural elements of BSS models and checking these graph con-

straints on the underlying abstract syntax graphs of BSS models in a similar way as in the case for business process models in Ch. 4.

Chapter 3

Behaviour Analysis of Visual Languages Based on Graph Transformation

Visual languages play an important role for software and system modelling. Especially the concept of domain specific modelling languages, i.e. the specification of modelling languages for particular domains and applications, shows a high impact in many application areas. In order to ensure correctness of the developed models with respect to given requirements there is a strong need for automated formal tool support. This section presents formal techniques for the analysis of visual behaviour models concerning the problems how to check whether two executions of the system are equivalent, how to generate all the executions that are equivalent to a given one and, moreover, how to check whether some modifications of the the order of steps of a given execution preserves the equivalence.

The analysis techniques are based on the formal framework of algebraic graph transformation used for the specification of operational semantics of visual models as it is used for the case studies of this thesis. All results, however, are shown to hold for general $\mathcal{M}$-adhesive transformation systems. For this reason, the result can be applied as well for a large variety of specification techniques including various kinds of graph transformation systems and different kinds of Petri net transformation systems.

Main concepts and constructions for the specification of visual models and its operational semantics are presented in Sec. 3.1, which allow for automated generation of sytax directed editor environments based on the Eclipse environment [Ecl10]. Sec. 3.2 introduces the notion of permutation equivalence, which is shown to be the adequate equivalence relation for transformation systems which use the intuitive and common concept of negative application conditions (NACs) for restricting the applicability of the rules of the operational semantics [Her09a]. Permutation equivalence is based on the standard notion of switch equivalence without NACs. By definition, two transformation sequences are permutation-equivalent, if they are switch equivalent without NACs and moreover, they respect all NACs. As shown by an intuitive example a direct analysis according to this

definition is complex and for this reason we show by Secs. 3.3 to 3.5 how the analysis can be performed on the generated compact process model. Sec. 3.3 presents the generation of the process model given by a subobject transformation system (STS) based on the results in [CHS08, Her08a, Her08b]. Thereafter, according to [Her09a], Sec. 3.4 presents the analysis of permutation equivalence based on STSs and shows as the first main result by Thm. 3.4.12 that the analysis is sound and complete with respect to interleaving semantics, i.e. the set of generated sequences specifies exactly the set of all permutation-equivalent transformation sequences to the given one. Finally, Sec. 3.5 shows that – independent of the chosen $\mathcal{M}$-adhesive category on which the transformation system is based – the analysis of permutation equivalence can be performed by generating a low-level P/T Petri net, called dependency net [HCEK10a]. This generation uses the derived STS and we show by Thm. 3.5.3 that the analysis is as well sound and complete. The advantage of the dependency net is that only the dependencies between the steps of the given transformation are specified leaving out the concrete details of the internal structure of the involved objects.

3.1 Specification of Visual Languages Based on Graph Transformation

Visual modelling is a fundamental concept in software and system modelling and visual models are increasingly developed and applied in many different areas. Compared to one-dimensional plain textual specifications visual models are multi-dimensional and show main advantages concerning abstraction and intuitive understanding. They provide additional diagrammatic elements, like boxes, connections and containers of different shapes. These extensions lead to an increased complexity of the specification techniques for defining visual languages compared to textual ones. Furthermore, the different application domains of visual languages require different features and properties of visual languages leading to the concept of domain specific modelling, i.e. the specification of domain specific modelling languages (DSLs) that are optimized for the particular domain. For this purpose, specification techniques for visual languages have to be powerful with respect to complexity and they have to provide a general and well founded specification framework.

The two major lines in the specification of visual languages are – on the one hand – meta modelling based on the Meta Object Facility (MOF [MOF06]) developed by OMG and – on the other hand – graph grammars that extend the concepts for string grammars to the case graphs. In both approaches the definition of the abstract syntax builds the main part of the specification. The concrete syntax of the models is aligned to the abstract syntax graphs and may contain some additional information for visualization that are not dependent on the abstract syntax directly.

In software design, the Unified Modelling Language (UML [OMG10]) developed and maintained by OMG has become a standard and its specification is also the reference for metamodelling with MOF. Various kinds of structure, behaviour and component diagram types are provided, which can be furthermore customized for particular domains using the concept of stereotypes. However, DSLs often are very specific to the application domain such that UML alone is not sufficient and metamodelling based on MOF allows to specify further visual languages.

The basis of metamodelling is the specification of a meta model, which generally specifies the possible structural elements of the abstract syntax of models. A meta model is given by a restricted form of UML class diagram mainly providing the notion of classes, attributes, associations, multiplicities and inheritance. In addition to the meta model the object constraint language (OCL [OCL03]) is used to further restrict the abstract syntax of models by additional constraints that have to be satisfied. The abstract syntax of the specified visual language VL consists of all possible models that conform to the meta model and additionally satisfy the OCL constraints. Therefore, metamodelling is a descriptive specification technique and does not directly provide a technique for generating valid models of the language.

Visual language definition based on graph transformation [EEPT06, Erm09] starts with the specification of a type graph TG, which is a similar concept to defining a meta model. Each model of the language is typed over TG which specifies the possible node types (corresponding to classes in MOF), edge types (corresponding to associations in MOF) and allows the specification of attribute types, multiplicities and inheritance similar to the corresponding concepts in MOF. Instead of defining additional constraints for specifying well-formedness conditions of models a graph grammar provides a start graph and construction rules. These graph transformation rules are used to create the valid visual models such that the graph grammar approach to visual language specification is constructive.

Graph transformation additionally provides the concept of graph constraints [EEPT06, HP09] as a descriptive component of visual language definition offering the power of first order logic on graphs [Ren04a]. The designer of a visual language is therefore free to either specify the type graph with constraints, the type graph with start graph and rules or a combination of both. From the formal point of view all three variants are considered to form a graph grammar, because the set of rules can be empty.

As mentioned before, the concrete syntax of a visual language is defined as an extension of the abstract syntax or, alternatively, separately from the abstract syntax using an additional mapping model to specify the correspondences. In the first case, the concrete syntax elements are attached directly to the corresponding abstract syntax elements, i.e. the type graph and rules are extended by additional elements that specify certain positions and figures for the visualization of the abstract syntax. The Tiger environment [EEHT05, Tig10, EE08b, BEEH08, BEEH09] is based on this approach and provides a

generator for visual editors as Eclipse plugins using directly a given language specification. In order to increase usability additional complex editing operations can be specified based on graph transformation rules and control structures. In the second case, the alignment of the abstract and concrete syntax can be described by plain text and suitable example visualizations as e.g. in the specification of UML. A more precise mapping model is defined using the Eclipse plugin GMF [GMF07], where corresponding abstract and concrete syntax elements are pairwise related. In [Tae06, TCSE08], GMF was extended by a component for the specification of graph transformation rules, such that also complex editing operations can be specified. In the following, we follow the graph grammar based approach for the specification of visual languages.

The analysis of visual behaviour models – as being the main focus of this chapter – is based on the definition of a semantics for a given visual language and its syntax definition. Besides the wide range of approaches for the specification of semantics of behavioural models mainly based on denotational semantics [Sto77, Sch86, Win93], graph transformation provides concepts for an intuitive specification of the formal semantics of behavioural models by the definition of graph transformation rules for the operational semantics. These graph transformation rules are defined on an extended type graph of the visual language specification. This integrated concept of syntax and semantics definition based on graph transformation further allows for simulation and animation [Erm06], which can be used to visualize analysis results. Furthermore, the algebraic approach to graph transformation provides a rich formal foundation [EPS73, Roz97, EEKR99, EKMR99, EEPT06] including powerful results concerning the analysis of graph transformation systems, like the Local Church-Rosser, Parallelism, Concurrency, Extension and Embedding Theorems. These results build the basis for the new techniques on behaviour analysis of visual models in this chapter and the further techniques in the chapters thereafter.

At first, we review the main constructions and results for the algebraic approach to typed attributed graph transformation [EEPT06], where graphs can be interpreted as algebras over a graph structure signature and a data signature. Plain graphs, i.e. graphs without attribution, are given by a set of nodes, a set of edges and two functions defining the source and target nodes of each edge. Hence, graphs may contain multiple edges between two edges and in the case of typed graphs these multiple edges may also have the same type.

Definition 3.1.1 (Graph and Graph Morphism)**.** *A* graph $G = (V, E, s, t)$ *consists of a set* V *of nodes (also called vertices), a set* E *of edges, and two functions* $s, t : E \to V$ *, the source and target functions:* $V \overset{s}{\underset{t}{\rightrightarrows}} E$.

Given graphs G_1, G_2 *with* $G_i = (V_i, E_i, s_i, t_i)$ *for* $i = 1, 2$*, a* graph morphism $f : G_1 \to G_2$ *,* $f = (f_V, f_E)$ *consists of two functions* $f_V : V_1 \to V_2$ *and* $f_E : E_1 \to E_2$ *that preserve the source and target functions, i.e.* $f_V \circ s_1 = s_2 \circ f_E$ *and* $f_V \circ t_1 = t_2 \circ f_E$*:*

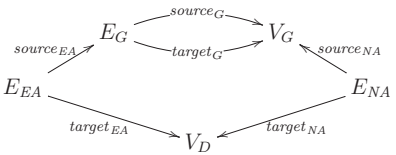
$$\begin{array}{ccc} E_1 & \overset{s_1}{\underset{t_1}{\rightrightarrows}} & V_1 \\ {\scriptstyle f_E}\downarrow & (=) & \downarrow{\scriptstyle f_V} \\ E_2 & \overset{s_1}{\underset{t_1}{\rightrightarrows}} & V_2 \end{array}$$

A graph morphism f is injective (or surjective) if both functions f_V , f_E are injective (or surjective, respectively); f is called isomorphic if it is bijective, which means both injective and surjective. The class of all graphs as objects and of all graph morphisms forms the category **Graphs**.

The attribution of graphs is based on the concept of E-graphs, which extend plain graphs by two further sets attribution edges – one for node attribution (E_{NA}) and one for edge attribution (E_{EA}) – together with a set of data values (V_D) that can be assigned as attribute values.

Definition 3.1.2 (E-graph and E-graph morphism)**.** *An* E-graph G *with* $G = (V_G, V_D, E_G, E_{NA}, E_{EA}, (source_j, target_j)_{j\in\{G,NA,EA\}})$ *consists of the sets*

- V_G *and* V_D *, called the graph and data nodes (or vertices), respectively;*
- E_G, E_{NA} *, and* E_{EA} *called the graph, node attribute, and edge attribute edges, respectively; and the source and target functions*
- $source_G : E_G \to V_G, target_G : E_G \to V_G$ *for graph edges;*
- $source_{NA} : E_{NA} \to V_G, target_{NA} : E_{NA} \to V_D$ *for node attribute edges; and*
- $source_{EA} : E_{EA} \to E_G, target_{EA} : E_{EA} \to V_D$ *for edge attribute edges:*



Consider the E-graphs G^1 and G^2 with $G^k = (V_G^k, V_D^k, E_G^k, E_{NA}^k, E_{EA}^k, (source_j^k, target_j^k)_{j\in\{G,NA,EA\}})$ for $k = 1, 2$. An E-graph morphism $f : G^1 \to G^2$ is a tuple $(f_{V_G}, f_{V_D}, f_{E_G}, f_{E_{NA}}, f_{E_{EA}})$ with $f_{V_i} : V_i^1 \to V_i^2$ and $f_{E_j} : E_j^1 \to E_j^2$ for $i \in \{G, D\}$, $j \in \{G, NA, EA\}$ such that f commutes with all source and target functions, for example $f_{V_G} \circ source_G^1 = source_G^2 \circ f_{E_G}$.

E-graphs form the structural part of attributed graphs. In order to specify the carrier sets of attribute values that form the single set V_D and to define the operations for generation and manipulating data values we extend E-graphs by an additional data algebra D. The carrier sets D_s of D contain the data elements for each sort $s \in S$ according to a data signature $DSIG = (S_D, OP_D)$. These carrier sets are combined by disjoint union and form the set V_D of data elements.

Definition 3.1.3 (Attributed Graph and Attributed Graph Morphism)**.** *Let $DSIG = (S_D, OP_D)$ be a data signature with attribute value sorts $S'_D \subseteq S_D$. An attributed graph $AG = (G, D)$ consists of an E-graph G together with a $DSIG$-algebra D such that $\uplus_{s \in S'_D} D_S = V_D$.*

For two attributed graphs $AG^1 = (G^1, D^1)$ and $AG^2 = (G^2, D^2)$, an attributed graph morphism $f : AG^1 \to AG^2$ is a pair $f = (f_G, f_D)$ with an E-graph morphism $f_G : G^1 \to G^2$ and an algebra homomorphism $f_D : D^1 \to D^2$ such that (1) commutes for all $s \in S'_D$, where the vertical arrows are inclusions.

$$\begin{array}{ccc} D^1_s & \xrightarrow{f_{D,s}} & D^2_s \\ \downarrow & (1) & \downarrow \\ V^1_D & \xrightarrow{f_{G,V_D}} & V^2_D \end{array}$$

Remark 3.1.4 (Further Attribution Concepts)**.** *The general concept of E-graphs can be used for further kinds of attribution like e.g. for constraint graphs [NHOH10], where attribute values are variables that are further specified by some formula. This concept allows to postpone concrete attribute computations leading to a lazy evaluation concept, which is of main importance for applications in service composition.*

Definition 3.1.5 (Category of Attributed Graphs)**.** *Given a data signature $DSIG$ as above, attributed graphs and attributed graph morphisms form the category* **AGraphs**.

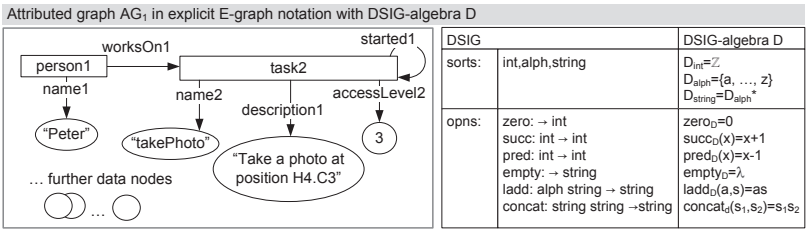


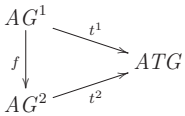
Figure 3.1: Example for an attributed graph including its data algebra

Example 3.1.6 (Attributed Graph)**.** *The attributed graph $AG_1 = (G_1, D)$ in Fig. 3.1 consists of an E-graph and a $DSIG$-data algebra D. The E-graph contains structural nodes (depicted as rectangles), structural edges between structural nodes, data nodes (depicted as ellipses) as well as edges for node attribution. The graph shows a person named "Peter", who is working on the task "takePhoto". However, this graph can also be interpreted completely different, because the structural nodes as well as the attribution edges are not typed and thus, their names have no formal semantic interpretation. For this reason, we extend this example to a typed attributed graph in Ex. 3.1.10.*

Typing is one further major concept for graph transformation systems besides attribution and further enriches the specification capabilities of the framework. Typing is used on the

one hand to ensure certain well-formedness constraints for all graphs that can be generated by a transformation system and on the other hand, to add information to each graph element for semantic interpretation. In the simplest case types are assigned as labels to each node and edge of a graph. The definition of a type graph additionally ensures that graphs typed over the type graph satisfy structural properties. This means in particular that the type graph specifies the types of the source and target nodes of each edge type ensuring that each edge of this type is always connected to source and target nodes of the specified types in the type graph. The relation of a typed graph towards its type graph is specified by a type morphism from the typed graph to the type graph. This morphism specifies the types for all nodes, edges, and attributes occurring in the typed graph. The data signature of the type graph already distinguishes the sorts of the carrier sets for the attribute values. For this reason, it suffices to define the data algebra of the type graph as final algebra over the data signature, i.e. each carrier set exactly contains the element that specifies the data type for the sort.

Definition 3.1.7 (Typed Attributed Graph and Typed Attributed Graph Morphism)**.** *Given a data signature $DSIG$, an attributed type graph is an attributed graph $ATG = (TG, Z)$, where Z is the final $DSIG$-algebra. A typed attributed graph (AG, t) over ATG consists of an attributed graph AG together with an attributed graph morphism $t : AG \to ATG$. A typed attributed graph morphism $f : (AG^1, t^1) \to (AG^2, t^2)$ is an attributed graph morphism $f : AG^1 \to AG^2$ such that $t^2 \circ f = t^1$:*



Definition 3.1.8 (Category of Typed Attributed Graphs)**.** *Typed attributed graphs over an attributed type graph ATG and typed attributed graph morphisms form the category* $\mathbf{AGraphs}_{ATG}$.

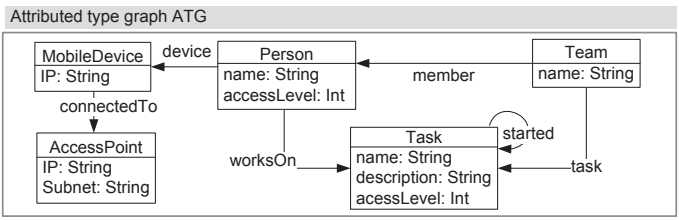


Figure 3.2: Attributed type graph for workflows mobile networks

Example 3.1.9 (Attributed Type Graph)**.** *Fig. 3.2 shows the type graph of an attributed graph grammar for modelling a workflow system in mobile ad hoc networks, where persons can be assigned to teams and tasks and they can change their location implying that their mobile communication devices may need to reconnect to new access points. Each team can assign its tasks to its team members, where each task can be performed by a group of team members and not necessarily by a single person. During the execution of a task the flag "started" will be attached to the task to specify that the task was started and not yet finished. The type graph is shown in compact notation, which is similar to the MOF notation for meta models [MOF06]. Attributes are depicted underneath the node names and the attribute types are placed directly behind the names of the attributes. This means that the attribution edges of underlying E-graph are implicit.*

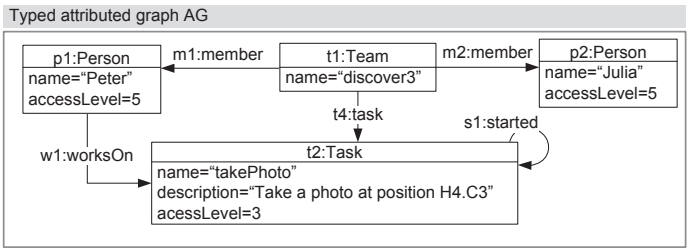


Figure 3.3: Typed attributed graph AG in compact notation

Example 3.1.10 (Typed Attributed Graph)**.** *An example of an attributed graph AG typed over the type graph ATG of Fig. 3.2 is shown in Fig. 3.3 and using the explicit E-graph notation in Fig. 3.4. It specifies a team with two team members ("Peter" and "Julia") and one task, which is currently performed by person $p1$ ("Peter"). Graph AG is shown in both, in compact notation (upper part) and in explicit E-graph notation (lower part). All data value nodes in the E-graph notation are visualized by ellipses and the dots and circles in the lower right part of the figure show that the further data nodes of the E-graph, which are not connected to the main part of the graph, are left implicit. Note further that both persons have an access level of "5" and thus, they share the same data value node. Finally, the explicit E-graph notation shows that formally, a structural node may also have multiple data values assigned to it, because there may be more than one edge outgoing with the same type. While this effect will usually not appear during transformation sequences, this generality of attribution is explicitly used in Sec. 3.3 for analysing the interleaving behaviour of transformation sequences.*

For more features of typing, such as the definition of multiplicities and node type inheritance, we refer to [EEPT06, LBE+07, EEH09]. The used multiplicities in our examples

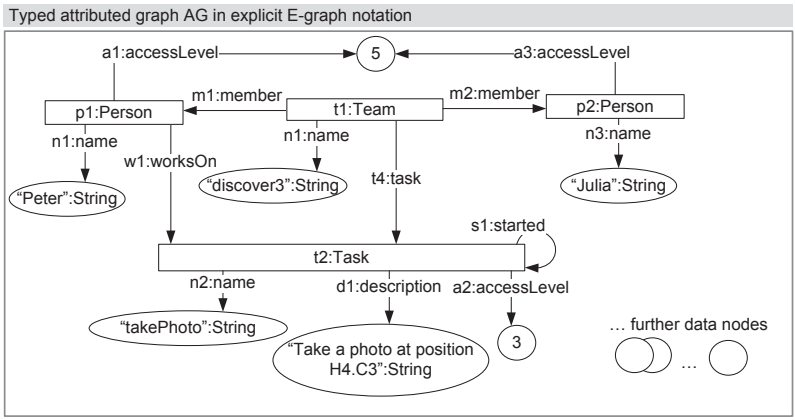


Figure 3.4: Typed attributed graph AG in explicit E-graph notation

will be ensured by the graph transformation rules and thus, they do not need to be checked separately. Furthermore, each multiplicity constraint can be transformed into application conditions for the graph transformation rules [EEPT06], which ensure that the multiplicities are respected during the rule application.

The category of typed attributed graphs is one prominent instantiation of the framework of $\mathcal{M}$-adhesive categories, which abstracts away some of the construction details while preserving all necessary properties needed to show the main results for transformation systems mentioned before. This abstract framework on the one hand simplifies several proofs of results on constructions for $\mathcal{M}$-adhesive categories and on the other hand, automatically induces the shown results to all possible instantiations of $\mathcal{M}$-adhesive categories, like hypergraphs, elementary Petri nets, place/transition Petri nets and algebraic high-level nets with and without individual tokens. For this reason, we will base all further constructions on the abstract framework of $\mathcal{M}$-adhesive categories and demonstrate them for typed attributed graph transformation systems.

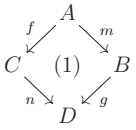
$\mathcal{M}$-adhesive categories fulfil the following three conditions, which are less restrictive than the conditions for adhesive categories [LS04], weak adhesive high level replacement (HLR) categories [LS04, EEPT06] and partial map adhesive categories [Hei10] as presented in [EGH10]. Thus, the notion of $\mathcal{M}$-adhesive categories is a generalization of the mentioned variants of adhesive categories.

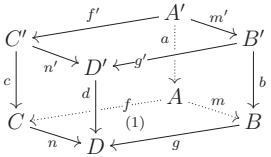
Definition 3.1.11 ($\mathcal{M}$-adhesive category)**.** *A pair* $(\mathbf{C}, \mathcal{M})$ *consisting of a category* $\mathbf{C}$ *and a class of morphism* $\mathcal{M}$ *is called an* $\mathcal{M}$-adhesive category *if:*

1. *$\mathcal{M}$ is a class of monomorphisms of* $\mathbf{C}$ *closed under isomorphisms, composition, and decomposition* $(g \circ f \in \mathcal{M}, g \in \mathcal{M} \Rightarrow f \in \mathcal{M})$.

2. $\mathbf{C}$ *has pushouts and pullbacks along $\mathcal{M}$-morphisms, and $\mathcal{M}$-morphisms are closed under pushouts and pullbacks.*

3. *Pushouts in* $\mathbf{C}$ *along $\mathcal{M}$-morphisms are* "$\mathcal{M}$ Van Kampen" ($\mathcal{M}$-VK) squares*, i.e. for any commutative cube like (2) below where the bottom face (1) is a pushout along $m \in \mathcal{M}$, the back faces are pullbacks, and $b, c, d \in \mathcal{M}$, we have: The top face is a pushout if and only if the front faces are pullbacks.*





(2)

Fact 3.1.12 ($(\mathbf{AGraphs}_{ATG}, \mathcal{M})$ is $\mathcal{M}$-adhesive)**.** *The category* $(\mathbf{AGraphs}, \mathcal{M})$ *of typed attributed graphs with class $\mathcal{M}$ of monomorphisms that are isomorphisms on the data part are $\mathcal{M}$-adhesive.*

Proof. By Thm. 11.11 in [EEPT06] the category is an adhesive HLR category. The conditions of adhesive HLR categories directly imply the conditions of $\mathcal{M}$-adhesive categories. Conditions (1) and (2) are equal and condition (3) for adhesive HLR categories has weaker preconditions. Thus, $(\mathbf{AGraphs}_{ATG}, \mathcal{M})$ is an $\mathcal{M}$-adhesive category. □

As motivated before, the operational semantics of visual behaviour models can be defined by operation graph transformation rules – also called graph productions. General transformation rules of arbitrary categories are defined in Def. 3.1.13 below including the concept of negative application conditions (NACs). A NAC prevents the application of a rule in cases when certain forbidden (negative) patterns are present at the matched part where the rule shall be applied. Graph transformation rules are, in general, attributed with a term algebra $T_{OP}(X)$ over the data signature $DSIG$ defined in the type graph. This allows for a flexible specification of attribute values using variables and terms over variables as demonstrated in Ex. 3.1.14.

Definition 3.1.13 (Transformation Rule with NACs)**.** *A* transforamtion rule *(also called production)* $p = (L_p \overset{l}{\hookleftarrow} K_p \overset{r}{\hookrightarrow} R_p)$ *is a pair of monomorphisms and we write* rule *for short. Given a rule p, then a* Negative Application Condition (NAC) *for p is a morphism $n : L_p \to N$, having the left-hand side of p as source. A rule p_0 together with a finite set of NACs $\mathbf{N}$ for p is called a* rule with NACs *p and specified as the pair $p = (p_0, \mathbf{N})$.*

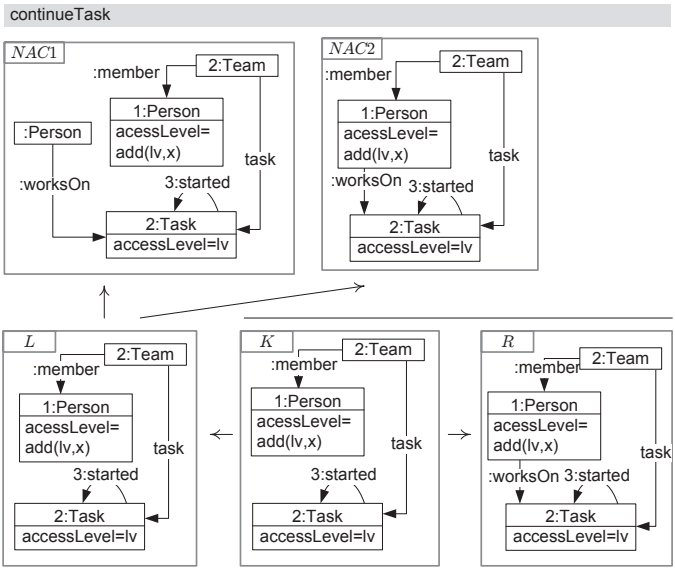
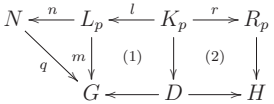


Figure 3.5: Graph transformation rule with NACs

Example 3.1.14 (Graph transformation rule with NACs)**.** *The rule "$continueTask$" shown in Fig. 3.5 assigns a new task to a person, who has an access level equal or above the specified access level of the task. This condition is ensured by the term "$add(lv, x)$" as attribute value for node "1". The two NACs ensure that the chosen task was not assigned to a person before and still is assigned. While the first NAC ensures that there is no other person assigned to the task the second NAC ensures that also the chosen person is not currently assigned to task "2".*

The application of a rule to a given object of a category is defined as double pushout according to Def. 3.1.15 below. The left hand side of the rule is matched into the given object G via a morphism $m : L_p \to G$. A NAC of a rule is violated, if the image of L_p via match m in G can be extended to an image of the "forbidden context" N. If all NACs are satisfied and the rule is applicable at m the first step is performed by constructing a pushout complement meaning for graphs that all elements that are in L but not in K are deleted leading to the intermediate graph D. Thereafter, the second step is constructed as pushout meaning for graphs that all nodes and edges that are additionally in R but are not present already in K are created leading to the resulting graph H.

Definition 3.1.15 (Transformation Step)**.** *Given a rule* $p = (p_0, \mathbf{N})$ *and a morphism* $m : L_p \to G$ *in an object* G*, called* match*. The match* m satisfies a NAC $n : L_p \to N$ for p*, written* $m \models n$*, if there is no monomorphism* $q : N \to G$ *such that* $q \circ n = m$*. A transformation step* $G \overset{p,m}{\Longrightarrow} H$ *from an object* G *to* H *using the rule* p *via match* m *exists, if (a) there are two pushouts (1) and (2), as depicted below; and (b)* $m \models n$ *for each NAC* $(n : L_p \hookrightarrow N) \in \mathbf{N}$*. We say that there is a* transformation step respecting NACs *If condition (a) above is satisfied (and (b) possibly not, thus NACs are ignored) we say that there is a* transformation step *from* G *to* H*. In both cases we write* $G \overset{p,m}{\Longrightarrow} H$*.*



For $\mathcal{M}$-adhesive categories we further have that pushout complements are unique up to isomorphism, if they exist, and thus, each transformation step is uniquely defined, because pushouts are unique up to isomorphism by definition. Since pushout complements do not exist in general, the existence has to be checked before applying a rule using the gluing condition. For graphs, the gluing condition is given by $GP \subseteq DP \cup IP$, i.e. the dangling and identification condition are satisfied. The dangling condition ensures that a node of a graph G may only be deleted if all adjacent edges in G are deleted as well by the rule. By requiring the identification condition the rule may only be applied, if nodes or edges that are identified by the match have to be preserved by the rule.

Remark 3.1.16 (NAC schemata for $(\mathbf{AGraphs}_{ATG}, \mathcal{M})$)**.** *The explicit specification of NACs in a typed attributed graph transformation system can be complex. Consider a match* $m : L \to G$ *and a NAC* $n : L \to N$*, where* L *is attributed via the term algebra* $T_{OP}(X)$*. If the data algebra* A_N *of* N *is not isomorphic to the data algebra* A_G *of* G*, then we have trivially* $m \models n$*, because if the existence of a compatible* $(q : N \to G) \in \mathcal{M}$ *implies* $A_N \cong A_G$*. Therefore,* A_N *has to be isomorphic to* A_G*, which means that the NAC is relevant for one particular variable assignment* $ass : X \to A_G$ *only and we would need to explicitly specify each of them.*

For this reason, we presented in Sec. 3.2 in [KHM06], how a single NAC (NAC schema) can represent the set of all possible variations via the possible variable assignments. For checking the NAC schema, the given match is used to derive the unique relevant concrete NAC N'*. More precisely, NAC-satisfaction requires in this case that there is no compatible* $\mathcal{M}$*-morphism* $q : N' \to G$ *with* N' *being obtained from* N *by a pushout of* $n : L \to N$ *and* $L \to m(L)$*, i.e. by performing the same identifications as in the match* $m : L \to G$*. This approach is also implemented in the tool AGG [AGG10], where each NAC is interpreted as NAC schema, which therefore allows for compact specifications of NACs.*

A set of graph transformation rules with assigned rule names together with a common type graph form a transformation system. Transformation systems play a big role for the

analysis of behaviour models and graph transformation systems, in particular, have been applied for analysing and simulating a wide range of visual behaviour models.

Definition 3.1.17 (Transformation System)**.** *A* transformation system (TS) with NACs over a category $\mathbf{C}$ *is a pair* $\mathrm{TS} = (TG, Q, \pi_N)$ *where* Q *is a set of rule names, and* π_N *maps each name* $q \in Q$ *to a rule with NACs* $\pi_N(q) = \langle \pi(q), \mathbf{N}_q \rangle$*. A* transformation sequence (respecting NACs) *of* TS *is a sequence* $G_0 \overset{q_1,m_1}{\Longrightarrow} G_1 \cdots \overset{q_n,m_n}{\Longrightarrow} G_n$*, where* $q_1, \ldots, q_n \in Q$ *and* $d_i = G_{i-1} \overset{\pi(q_i),m_i}{\Longrightarrow} G_i$ *are transformation steps (respecting NACs) for* $i \in 1, \ldots, n$*. Sometimes we denote a transformation sequence as a sequence* $d = (d_1; \ldots; d_n)$ *of transformation steps.*

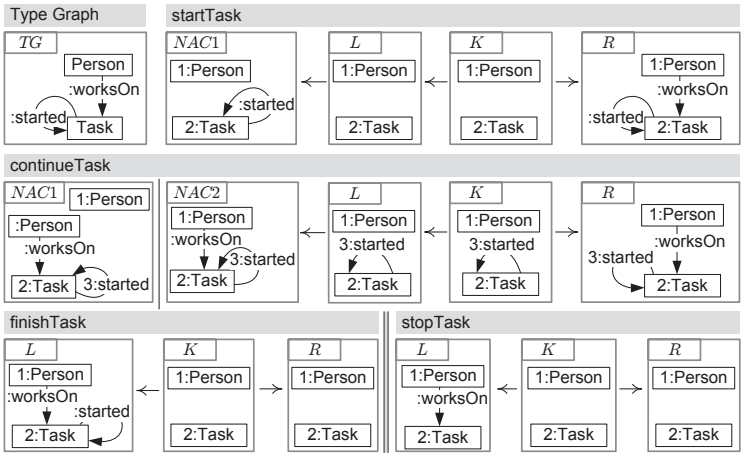


Figure 3.6: Reduced transformation system GS as running example

Example 3.1.18 (Graph Transformation System with NACs)**.** *Fig. 3.6 shows a part of an attributed graph transformation system for modelling a workflow system in mobile ad hoc networks, where persons can be assigned to teams and tasks and they can change their location implying that their mobile communication devices may need to reconnect to new access points. The* type graph TG *shows that nodes in the system represent either persons or tasks: a task is active if it has a ":started" loop, and it can be assigned to a person with a ":worksOn" edge. Rule "stopTask" cancels the assignment of a task to a person; rule "continueTask" instead assigns the task, and it has two NACs to ensure that the task is not assigned to a person already. Fig. 3.7 shows two transformation sequences respecting NACs of* GS*. In transformation sequence* d *the only task is first continued*

by "1:Person", and then, after being stopped, by "2:Person". In d' the roles of the two Persons are inverted.

Based on a given transformation sequence of a transformation system we define in Secs. 3.2 - 3.5 techniques for analysing equivalence of transformation sequences. These techniques are used in the further chapters for applications to enterprise models and model transformations.

3.2 Behaviour Analysis Based on Switch and Permutation Equivalence

As introduced and presented in Sec. 3.1 before, $\mathcal{M}$-adhesive transformation systems with negative application conditions (NACs) [HHT96, EEPT06] are a suitable modelling framework for several application domains, and in particular for the specification of operational semantics of visual behaviour models. In this context, the analysis of concurrent and interleaving behaviour of the modelled system is of major interest. In order to provide suitable behaviour analysis techniques in Secs. 3.3 to 3.5 this section presents the general notion of permutation equivalence, introduced in [Her09a], on which the analysis techniques in the subsequent sections are based.

Switch equivalence has been successfully used in many domains for the analysis of concurrent behaviour. Intuitively, two executions of a system are switch-equivalent, if the can be transformed into each other by switching independent neighbouring execution steps. This means that the dependency analysis of neighbouring steps is the fundamental part of the overall equivalence analysis. In the context of visual behaviour models and their operational semantics specified by a transformation systems the concept of Negative Application Conditions (NACs) is widely used, because the application of each particular transformation rule can be prohibited at certain states of the system, where the system is not allowed to perform certain steps. The additional specification feature of NACs introduces new dependencies which increase the complexity of a complete and sound analysis of equivalence for transformation sequences. Indeed, the notion of switch equivalence with NACs is too strict, because there are intuitively equivalent transformation sequences which are not switch-equivalent if NACs are considered. For this reason we propose the notion of permutation equivalence as the most general extension of switch equivalence that still ensures NAC consistency of all derived equivalent transformation sequences to a given one. By definition, two transformation sequences are permutation-equivalent, if they respect the NACs and disregarding the NACs they are switch-equivalent.

Using the introduced case study in Sec. 3.1 this section show that permutation equivalence leads to all intuitively equivalent transformation sequences and that an analysis of

permutation equivalence directly based on the definition will cause high complexity in general. For this reason, we show in subsequent Secs. 3.3 to 3.5 how an analysis can be efficiently performed by constructing first a process model and, if required, transforming the first process model into a reduced process model which still allows for a complete analysis of permutation equivalence. This way, the efficiency of the analysis is substantially improved and the executable process models can be constructed in polynomial time.

In the following example, we present two intuitively equivalent transformation sequences of the transformation system – introduced in Sec. 3.1 before – and explain why these transformation sequences are not switch-equivalent. Thereafter, we present the formal notions which are used for the definition of switch equivalence and moreover, for the definition of permutation equivalence and show that the example transformation sequences are permutation-equivalent.

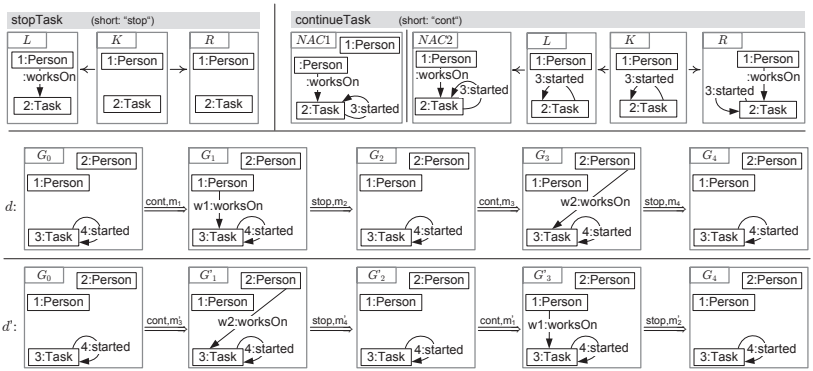


Figure 3.7: Transformation sequence d of GS and permutation-equivalent transformation sequence d'

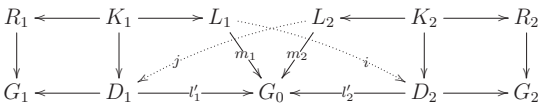
Example 3.2.1 (Equivalent Transformation Sequences)**.** *The two transformation sequences shown in Fig. 3.7 specify two executions within the modelled mobile ad hoc network for which the operation semantic rules are presented in Ex. 3.1.18 in Sec. 3.1. The transformation sequences are kept as simple and compact as possible to put the main focus on the formal aspects of the analysis of switch equivalence and permutation equivalence. The start state defined by the graph G_0 shows two persons and one task which was already started but there is currently no person working on this task. Hence, either person "1" or person "2" can continue the task, but not both at the same time, because the system has to ensure exclusive exections specified by the NACs of the operational semantics rule "continueTask" in Ex. 3.1.18. In the first step of transformation sequence d the first person continues the task (using rule "continueTask" abbreviated by "cont") and in transformation sequence*

d' the task is continued by the second person. In both transformation sequences the assigned persons stop again their work in the second step (via rule "stopTask" abbreviated by "stop") and the transformation sequences are continued by the assignment of the task to the corresponding other person, who finally stops the work as well without finishing the task.

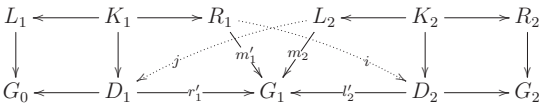
Now, both transformation sequences consist intuitively of the same transformation steps applied at equivalent matches, i.e. the left hand sides of rules in corresponding steps are mapped at the same elements in of the instance graphs. The only difference is the order in which the steps are executed. Thus, the transformation sequences d and d' are intuitively equivalent. However, switch equivalence does not relate them. Each pair of neighbouring steps is dependent, such that no switching is possible. Considering transformation sequence d we have that the second step depends on the first one because it requires the edge "w1" to be present in order to delete it. The third step forbids an edge of type "worksOn" to be present causing the third step to depend on the second one. Finally, the forth step requires the edge "w2" to be present in order to delete it, i.e. it depends on the third step. A similar argumentation holds for transformation sequence d'.

The classical theory of the DPO approach (without NACs) introduces switch equivalence among transformation sequences as an equivalence which relates transformation sequences that differ only in the order in which independent transformation steps are performed (see [Kre86, BCH+06]). This concept of *switch equivalence* is based on the notion of *sequential independence* and on the *Local Church-Rosser theorem*. In the next definitions we summarise the corresponding formal conditions and constructions.

Definition 3.2.2 (Parallel and Sequential Independence of Transformation Sequences Without NACs)**.** *Given two transformation steps $d_1 = (G_0 \overset{p_1,m_1}{\Longrightarrow} G_1)$ and $d_2 = (G_0 \overset{p_2,m_2}{\Longrightarrow} G_2)$. They are* parallel independent *if there exist arrows $i : L_1 \to D_2$ and $j : L_2 \to D_1$ such that $l'_2 \circ i = m_1$ and $l'_1 \circ j = m_2$.*



Let $d_1 = G_0 \overset{p_1,m_1}{\Longrightarrow} G_1$ and $d_2 = G_1 \overset{p_2,m_2}{\Longrightarrow} G_2$ be two transformation steps. Then they are sequentially independent *if there exist arrows $i : R_1 \to D_2$ and $j : L_2 \to D_1$ such that $l'_2 \circ i = m'_1$ and $r'_1 \circ j = m_2$.*



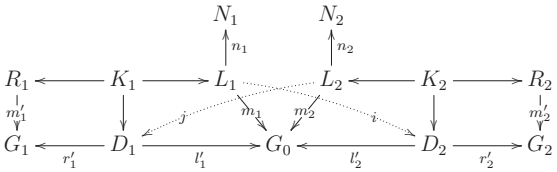
Switch equivalence relates transformation sequences which can be derived from each other by switching sequentially independent steps, i.e. neighbouring steps that do not de-

pend on each other. By the Local Church Rosser Theorem (Thm. 5.12 in [EEPT06]) we know that two sequentially independent steps $G_0 \overset{p_1,m_1}{\Longrightarrow} G_1 \overset{p_2,m_2}{\Longrightarrow} G_2$ can be switched leading to sequentially independent steps $G_0 \overset{p_2,m_2'}{\Longrightarrow} G_1' \overset{p_1,m_1'}{\Longrightarrow} G_2$ starting at the same object G_0 and ending at the same object G_2 using the same rules and equivalent matches. For transformation systems without NACs, switch equivalence leads to the complete set of its equivalent transformation sequences as shown in [BCH+06]. Since DPO transformation diagrams are unique up to isomorphism only we relate isomorphic transformation diagrams by "$\cong$" meaning that there are isomorphisms between the objects compatible with the involved morphisms. Intuitively, $d \cong d'$, if they have the same length and there are isomorphisms between the corresponding objects of d and d' compatible with the involved morphisms.

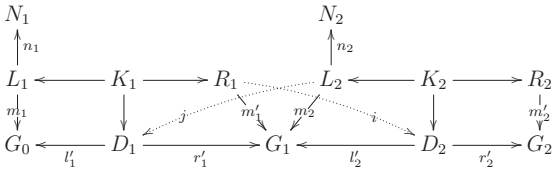
Definition 3.2.3 (Switch Equivalence Without NACs)**.** *If d_1 and d_2 are two sequentially independent transformation steps without NACs, then according to the Local Church Rosser Theorem (Thm. 5.12 in [EEPT06]) they can be "switched" obtaining the transformation steps $d_2' = G_0 \overset{p_2,\overline{m_2}}{\Longrightarrow} G_1'$ and $d_1' = G_1' \overset{p_1,\overline{m_1}}{\Longrightarrow} G_2$, which apply the two rules in the opposite order. Now, let $d = (d_1; \ldots; d_k; d_{k+1}; \ldots; d_n)$ be a transformation sequence, where d_k and d_{k+1} are two sequentially independent transformation steps without NACs, and let d' be obtained from d by switching them according to the Local Church Rosser Theorem. Then, d' is a* switching of d*, written $d \overset{sw}{\sim} d'$. The* switch equivalence*, denoted $\overset{sw}{\approx}$, is the smallest equivalence relation on transformation sequences containing both $\overset{sw}{\sim}$ and the relation $\cong$ for isomorphic transformation sequences.*

We now extend the notion of switch equivalence to transformation sequences with NACs using sequential independence for transformation sequences with NACs according to [HHT96, LEO06]. For this purpose, we use the local Church-Rosser Thmeorem for $\mathcal{M}$-adhesive transformation systems with NACs as shown by Thm. 3.4.3 in [Lam09] for weak adhesive HLR systems and the proof uses only conditions, which are also ensured for the slightly more general notion of $\mathcal{M}$-adhesive transformation systems. Note that the distinguished class $\mathcal{Q}$ in [Lam09] is considered in this thesis to coincides with $\mathcal{M}$, which is also the usual case.

Definition 3.2.4 (Parallel and Sequential Independence With NACs)**.** *Let $d_1 = (G_0 \overset{p_1,m_1}{\Longrightarrow} G_1)$ and $d_2 = (G_0 \overset{p_2,m_2}{\Longrightarrow} G_2)$ be two transformation steps with NACs, such that they are parallel independent without NACs, i.e. there exist arrows $i : L_1 \to D_2$ and $j : L_2 \to D_1$ such that $l_2' \circ i = m_1$ and $l_1' \circ j = m_2$. Let further $\overline{m}_1 = r_2' \circ i : L_1 \to G_2$ and $\overline{m}_2 = r_1' \circ j : L_2 \to G_1$ and let $\overline{m}_1, \overline{m}_2$ satisfy all NACs, i.e. $\overline{m}_2 \models n_2$ for each NAC $(n_2 : L_2 \to N_2)$ of p_2 and $\overline{m}_1 \models n_1$ for each NAC $(n_1 : L_1 \to N_1)$ of p_1. Then, $(G_0 \overset{p_1,m_1}{\Longrightarrow} G_1)$, $(G_1 \overset{p_2,m_2}{\Longrightarrow} G_2)$ are two* parallel independent transformation steps with NACs*.*



Let $d_1 = (G_0 \overset{p_1,m_1}{\Longrightarrow} G_1)$ *and* $d_2 = (G_1 \overset{p_2,m_2}{\Longrightarrow} G_2)$ *be two transformation steps with NACs, such that they are sequentially independent without NACs, i.e. there exist arrows* $i : R_1 \to D_2$ *and* $j : L_2 \to D_1$ *such that* $l'_2 \circ i = m'_1$ *and* $r'_1 \circ j = m_2$*. Let further* $\overline{m}_1 : L_1 \to G'_1$ *and* $\overline{m}_2 = l'_1 \circ j : L_2 \to G_0$ *be the derived matches by the Local Church Rosser Theorem (Thm. 3.4.3 in [Lam09]) and let* $\overline{m}_1, \overline{m}_2$ *satisfy all NACs, i.e.* $\overline{m}_2 \models n_2$ *for each NAC* $(n_2 : L_2 \to N_2)$ *of* p_2 *and* $\overline{m}_1 \models n_1$ *for each NAC* $(n_1 : L_1 \to N_1)$ *of* p_1*. Then,* $(G_0 \overset{p_1,m_1}{\Longrightarrow} G_1)$*,* $(G_1 \overset{p_2,m_2}{\Longrightarrow} G_2)$ *are two* sequentially independent transformation steps with NACs*.*



Switch equivalence with NACs is defined based on the notion of sequential independence with NACs and the Local Church Rosser Theorem with NACs (Thm. 3.4.3 in [Lam09]) and differs from switch equivalence without NACs in the way that less switchings are possible. Each switching has to respect the NACs and thus, a chain of switchings that violates some NACs but leads to a new transformation sequence that respects all NACs as in Ex. 3.2.1 is not possible.

Definition 3.2.5 (Switch Equivalence With NACs)**.** *If* d_1 *and* d_2 *are two sequentially independent transformation steps with NACs, then according to the Local Church Rosser Theorem they can be "switched" obtaining the transformation steps* $d'_2 = G_0 \overset{p_2,\overline{m}_2}{\Longrightarrow} G'_1$ *and* $d'_1 = G'_1 \overset{p_1,\overline{m}_1}{\Longrightarrow} G_2$*, which apply the two rules in the opposite order and respect all NACs. Now, let* $d = (d_1; \ldots; d_k; d_{k+1}; \ldots; d_n)$ *be a transformation sequence, where* d_k *and* d_{k+1} *are two sequentially independent transformation steps with NACs, and let* d' *be obtained from* d *by switching them according to the Local Church Rosser Theorem. Then,* d' *is a* switching of d*, written* $d \overset{sw}{\sim} d'$*. The* switch equivalence with NACs*, denoted* $\overset{swN}{\approx}$*, is the smallest equivalence relation on transformation sequences containing both* $\overset{swN}{\sim}$ *and the relation* $\cong$ *for isomorphic transformation sequences.*

The notion of switch equivalence with NACs does not identify all intuitively equivalent transformation sequences. The reason is that, in presence of NACs, there might be an equivalent permutation of a transformation sequence that cannot be derived by switch equivalence. Looking at d in Fig. 3.7 of Ex. 3.2.1 there is no pair of consecutive transformation steps which is sequentially independent if NACs are considered. However, the transformation sequence d' should be considered as equivalent. There are also examples in which even the switching of blocks of several steps would not lead to all permutation-equivalent transformation sequences. This brings us to the following, quite natural notion of permutation equivalence of transformation sequences respecting NACs, first proposed in [Her09a]. Note that for permutation-equivalent transformation sequences $d \overset{\pi}{\approx} d'$ the sequence of rules used in d' is a permutation of those used in d.

Definition 3.2.6 (Permutation Equivalence of Transformation Sequences)**.** *Two transformation sequences d and d' respecting NACs are* permutation equivalent*, written $d \overset{\pi}{\approx} d'$ if, disregarding the NACs, they are switch equivalent as for Def. 3.2.3.*

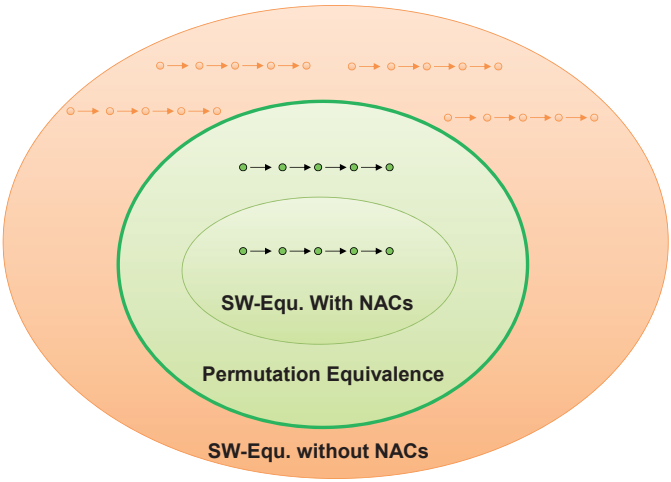


Figure 3.8: Hierarchy of Equivalenc Relations

By definition we have that two permutation-equivalent transformation sequences are switch-equivalent without NACs and furthermore, given two transformation sequences that are switch-equivalent with NACs then they are also permutation equivalent, because all NACs are respected. This leads to the hierarchy of the equivalence relations visualized in Fig. 3.8. Thus, for a given transformation sequence with NACs we have that the corresponding sets of equivalent transformation sequences using the three different notions form

a the total order via inclusions depicted in Fig. 3.8. In the special case that a transformation system does not contain NACs all sets are collapse to the same set, but if the transformation system contains NACs the relations usually differ quite heavily. Since switch equivalence is shown to induce the complete set of all equivalent transformation sequences if NACs are not considered, we have that permutation equivalence restricts this set to those transformation sequences that respect NACs. For this reason, permutation equivalence is the required relation that induces exactly all equivalent and NAC-consistent transformation sequences to a given transformation sequence with NACs.

While permutation equivalence is by definition a restriction of switch equivalence without NACs we now show that an analysis of permutation equivalence by generating first all switch equivalent transformation sequences without respecting NACs and then filtering out those with are not NAC consistent is naive and in general too inefficient for complex examples. For this reason, the subsequenct sections present alternative techniques that generate NAC consistent sequences only while still being complete, such that permutation equivalence can be analysed substantially more efficient.

In order to compare efforts for practical applications we extend the studied transformation sequences d and d' in Ex. 3.2.1 and compare the amounts of equivalent transformation sequences for the three considered equivalence relations, i.e. for switch equivalence without NACs, for permutation equivalence and for switch equivalence with NACs. This comparison shows that an analysis of permutation equivalence as described above should be avoided, because to many transformation sequences are generated that are not equivalent, meaning that the direct analysis would be far to complex for practical scenarios.

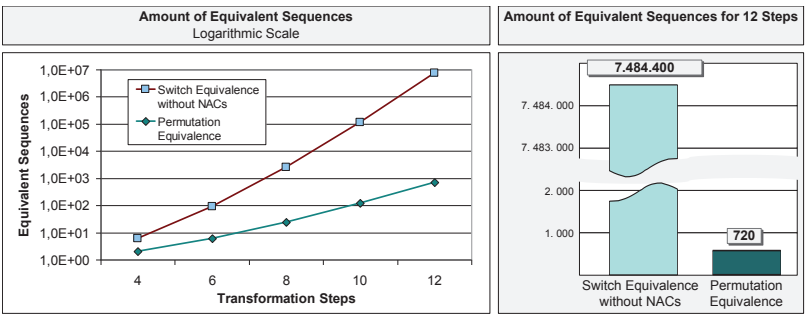


Figure 3.9: Comparison of the Amount of Equivalent Sequences

Example 3.2.7 (Extended Transformation Sequence)**.** *We extend the transformation sequence d of Ex. 3.2.1 to a transformation sequence $\widetilde{d}$, which specifies that the two persons are working on the same task, but they continue and stop their work three times, i.e.*

$\widetilde{d} = (d; d; d)$ is a transformation sequence with 12 steps. Note that still all steps in $\widetilde{d}$ are sequentially dependent with NACs and therefore, no direct switching respecting NACs is possible. Fig. 3.9 shows how the different amounts of equivalent sequences develop for 2 up to 6 blocks of "continue;stop" steps, i.e. starting with transformation sequence d with 2 blocks and ending at transformation sequence $\widetilde{d}$ with 6 blocks. Each permutation-equivalent transformation sequence of $\widetilde{d}$ has to preserve these blocks, otherwise a NAC or the causality relation (create-delete dependency) would be violated. Thus there are $6! = 720$ permutation-equivalent transformation sequences. The overall amount of transformation sequences that are switch-equivalent without NACs, however, reaches the number $12!/(2^6) = 7.484.400$ transformation sequences. This number is obtained by the following observation. For each permutation-equivalent sequence we can move the rule occurrences of the rule "stop" to later positions by switching sequentially independent neighbouring steps, because NACs are not considered for switch equivalence without NACs. Thus, the last (right most) rule occurrence of "stop" cannot be moved directly (=1 possible position), for the second last rule occurrence of "stop" there are 2 possible later positions (= 3 possible positions) and so forth. This means that we have $F = 1 \times 3 \times \cdots \times 11 = 10.395$ switch-equivalent sequences for each single permutation equivalent transformation sequences. This leads to a number of $6! \times F = 12!/2^6 = 7.484.400$ switch equivalent transformation sequences. Considering other sequences of iterated applications of d with n steps we have $(n/2)!$ permutation-equivalent transformation sequences and $n!/2^{(n/2)!}$ transformation sequences that are switch-equivalent without NACs. These numbers show that an analysis of permutation equivalence should not require to first generate all transformation sequences that are switch equivalent without NACs.

The comparison shows a significant difference of the amounts of equivalent sequences depending on the chosen notion of equivalence. This effect is not limited to the considered example, but appears in general if NACs are effectively involved in a transformation sequence, i.e. if intermediate graphs may violate them. Furthermore, a direct construction of the switch-equivalent transformation sequences without NACs is complex in addition. First of all, the amount of all possible permutations of the transformation steps is high in general and furthermore, the permutations have to be derived from the original transformation sequence by stepwise switching independent neighbouring steps. This includes the computation of the new matches and the new intermediate objects from the old ones. Finally, once all switch-equivalent transformation sequences without considering NACs are generated, the NACs have to be checked which again requires to perform the costly pattern matching process. The following sections will show how, on the one hand, the amount of generated sequences is reduced to the permutation-equivalent ones and, on the other hand, how the costs for pattern matching are heavily reduced.

3.3 Interleaving Semantics of $\mathcal{M}$-adhesive Transformation Systems

A process of a behaviour model specifies an equivalence class of executions, which differ only in the order of the executed steps. Thus, a process provides a compact model for the analysis of interleaving semantics concerning one concrete execution and its equivalent ones. This section presents the formal constructions and results for processes of arbitrary $\mathcal{M}$-adhesive transformation systems with respect to their interleaving semantics. The process construction yields a subobject transformation system (STS) which can be considered as a simplified variant of a DPO rewriting system which defines the possible equivalent transformation sequences to a given transformation sequence of the original transformation system. The results of this section are instantiated to the framework of typed attributed graph transformation systems in order to specify and analyse processes of visual behaviour models based on the operational semantics as presented in Sec. 3.1. This builds the basis for the analysis of interleaving semantics using the techniques in Secs. 3.4 and 3.5 based on the notion of permutation equivalence.

Processes of graph transformation systems based on the DPO approach are defined as occurrence grammars in [CMR96, Bal00]. In [BCH$^+$06], they are lifted to the abstract setting of adhesive rewriting systems in order to generalise the process construction, such that it can be instantiated to all adhesive rewriting systems. This opened possibilities for analysing processes of transformation systems based on arbitrary adhesive categories [LS04], such as typed graphs, graphs with scopes and graphs with second order edges.

In [CHS08], we introduced subobject transformation systems (STSs) as a novel formal framework for the analysis of transformation sequences of transformation systems based on the algebraic, double-pushout (DPO) approach. This setting allows for a direct analysis of all possible notions of dependency between any two productions without requiring any explicit match – matches are automatically given by the subobject relation. In particular, several equivalent characterizations of independence of productions are proposed, as well as a local Church-Rosser theorem in the setting of STSs. We further show how a given transformation sequence in an $\mathcal{M}$-adhesive transformation system leads to an STS via a suitable construction and show that relational reasoning in the resulting STS is sound and complete with respect to the independence in the original transformation sequence. For this purpose we extend the process construction to general matching and general $\mathcal{M}$-adhesive transformation systems that satisfy suitable conditions, which we presented in [Her08b]. Furthermore, as presented in [Her09a], we show how the construction is extended to systems with NACs. Our case study uses a typed attributed graph transformation system as one possible instantiation of the $\mathcal{M}$-adhesive framework.

Subobject transformation systems are based on the notion of subobjects. In the general case, a subobject A of an object T of a category $\mathbf{C}$ is an equivalence class of monomor-

phisms $a : A \to T$. We write A for short to denote the full equivalence class by a representative of the equivalence class and we leave the monomorphism a implicit. The category of subobjects of T is called $\mathbf{Sub}(T)$ and its morphisms $f : A \to B$ are those monomorphisms in $\mathbf{C}$, which are compatible with the implicit monomorphisms to T, i.e. $b \circ f = a$ for $a : A \to T$ and $b : B \to T$. If such an f exists, we write $A \subseteq B$ for short. Note in particular that if there is a morphism $f : A \to B$ between two subobjects A and B, then f is always unique. This implies that ($A \subseteq B$ and $B \subseteq A$) is equivalent to $A = B$ denoting that the equivalence class A is equal to the equivalence class B of subobjects and it is also equivalent to $f : A \to B$ being an isomorphism.

For arbitrary $\mathcal{M}$-adhesive categories we need to refine the notion of subobjects to $\mathcal{M}$-subobjects in order to ensure that all results and constructions for STSs can be performed. This means that instead of monomorphisms we consider $\mathcal{M}$-morphisms only and the category of $\mathcal{M}$-subobjects $\mathbf{Sub}_{\mathcal{M}}(T)$ contains as objects all equivalence classes of $\mathcal{M}$-morphisms $a : A \to T \in \mathcal{M}$ and as morphisms the compatible $\mathcal{M}$-morphisms. Note that in the case of adhesive categories we have the the class of $\mathcal{M}$-morphisms contains all monomorphisms and thus, the notion of $\mathcal{M}$-subobjects coincides with the notion of subobjects in this case.

Definition 3.3.1 (Category of $\mathcal{M}$-Subobjects)**.** *Let $\mathbf{C}$ be an $\mathcal{M}$-adhesive category and T be an object in $\mathbf{C}$. The category of $\mathcal{M}$-subobjects of $\mathbf{C}$ is called $\mathbf{Sub}_{\mathcal{M}}(T)$. Its objects are the subobjects $[a : A \to T]$ of T, where a is an $\mathcal{M}$-morphism. A morphism $f : A \to B$ in $\mathbf{Sub}_{\mathcal{M}}(T)$ is compatible with the implicit $\mathcal{M}$-morphisms, i.e. $b \circ f = a$, implying that f is an $\mathcal{M}$-morphism in $\mathbf{C}$ as well and we write $A \subseteq B$ for short.*

Notice that category $\mathbf{Sub}_{\mathcal{M}}(T)$ in general is not $\mathcal{M}$-adhesive, even if $\mathbf{C}$ is. In fact, let $A \subseteq B$ be an arrow in $\mathbf{Sub}_{\mathcal{M}}(T)$ which is not an isomorphism; then the pushout object of the span $B \supseteq A \subseteq B$ is easily shown to be B itself, but the resulting square is not a pullback, contradicting the fact that pushouts along $\mathcal{M}$-morphisms in an $\mathcal{M}$-adhesive category are pullbacks [EEPT06].

The following notions of "intersection" and "union" are the basic constructions for all further constructions, e.g. derivations in an STS, which can be constructed in a simpler way than transformation sequences in the underlying $\mathcal{M}$-adhesive transformation system.

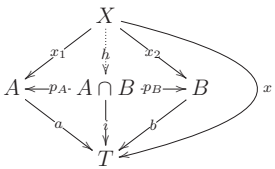
Definition 3.3.2 (Intersection and Union in $\mathbf{Sub}_{\mathcal{M}}(T)$)**.** *Let $\mathbf{C}$ be an $\mathcal{M}$-adhesive category and T be an object in $\mathbf{C}$. Let $\mathbf{Sub}_{\mathcal{M}}(T)$ be the category of $\mathcal{M}$-subobjects of T. The* intersection *$A \cap B$ of two objects A and B in $\mathbf{Sub}_{\mathcal{M}}(T)$ is the product in $\mathbf{Sub}_{\mathcal{M}}(T)$. The* union *$A \cup B$ of A and B in $\mathbf{Sub}_{\mathcal{M}}(T)$ is the coproduct in $\mathbf{Sub}_{\mathcal{M}}(T)$. The union of A and B is called* effective*, if $(A \to A \cup B \leftarrow B)$ is the pushout of $(A \leftarrow A \cap B \to B)$ in $\mathbf{C}$.*

We now characterize the constructions "intersection" and "union" for $\mathcal{M}$-subobjects in order to show the further properties - in particular the distributivity - which are needed for the construction and analysis of an STS of a given transformation sequence.

Lemma 3.3.3 (Intersection in $\mathbf{Sub}_{\mathcal{M}}(T)$)**.** *Let* $\mathbf{C}$ *be an* $\mathcal{M}$*-adhesive category and* T *be an object in* $\mathbf{C}$*. Let* $\mathbf{Sub}_{\mathcal{M}}(T)$ *be the category of* $\mathcal{M}$*-subobjects of* T*. The intersection* $A \cap B$ *of two subobjects* A *and* B *in* $\mathbf{Sub}_{\mathcal{M}}(T)$ *exists and is given by the pullback* (1) *in* $\mathbf{C}$ *with the* $\mathcal{M}$*-morphism* $i : A \cap B \xrightarrow{a \circ p_A} T$*.*

$$\begin{array}{ccc} A \cap B & \xrightarrow{p_A} & A \\ {\scriptstyle p_B}\downarrow & (1) & \downarrow{\scriptstyle a} \\ B & \xrightarrow[b]{} & T \end{array}$$

Proof. Let A, B be two objects in $\mathbf{Sub}_{\mathcal{M}}(T)$ and (1) be a pullback in $\mathbf{C}$. The pullback exists, because $a \in \mathcal{M}$. The projections p_A, p_B are in $\mathcal{M}$, because $a, b \in \mathcal{M}$ and $\mathcal{M}$ is closed under pullbacks. Furthermore, p_A, p_B are morphisms in $\mathbf{Sub}_{\mathcal{M}}(T)$ by the commutativity of the pullback construction and the definition of ab. Now, a comparison object $\mathbf{X}$ for the product $A \cap B$ in $\mathbf{Sub}_{\mathcal{M}}(T)$ is also a comparison object for the pullback $A \cap B$ in $\mathbf{C}$, because every diagram in $\mathbf{Sub}_{\mathcal{M}}(T)$ commutes. Thus, there is a unique morphism h satisfying the universal property of both, the pullback in $\mathbf{C}$ and the product in $\mathbf{Sub}_{\mathcal{M}}(T)$. Furthermore, $h \in \mathcal{M}$ by decomposition of x_1 and h is a morphism in $\mathbf{Sub}_{\mathcal{M}}(T)$ by the commutativity of the diagram beneath. □

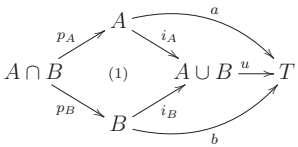


Remark 3.3.4 (Construction of Intersection and Union)**.** *The intersection and union of two subobjects are defined for the category* $\mathbf{Sub}_{\mathcal{M}}(T)$ *of* $\mathcal{M}$*-subobjects of* T *for a given* $\mathcal{M}$*-adhesive category* $(\mathbf{C}, \mathcal{M})$*. In many application domains, the constructions are required to be characterized by constructions in* $\mathbf{C}$ *itself. By Lem. 3.3.3 intersections in* $\mathbf{Sub}_{\mathcal{M}}(T)$ *can be always constructed as pullbacks over the super object* T *in* $\mathbf{C}$*. For unions, however, this result is not available. For the concrete category of typed attributed graphs* $(\mathbf{AGraphs}_{\mathbf{ATG}}, \mathcal{M})$*, we show by Thm. 3.3.5 that the union of two subobjects in* $\mathbf{Sub}_{\mathcal{M}}(T)$ *can be constructed as the pushout over the intersection. As the proof shows, this property can be usually easy shown for several relevant* $\mathcal{M}$*-adhesive categories and if the class* $\mathcal{M}$ *contains all monomorphisms, then the result is immediate using Thm. 4.7 in [LS04].*

We now show that the $\mathcal{M}$-adhesive category $\mathbf{AGraphs}_{\mathbf{ATG}}$ has effective unions, i.e. they can be constructed within $\mathbf{AGraphs}_{\mathbf{ATG}}$ as the pushout over the intersection in $\mathbf{Sub}_{\mathcal{M}}(T)$, i.e. $\mathbf{AGraphs}_{\mathbf{ATG}}$. The main part here is to show that the constructed object is again a subobject in $\mathbf{Sub}_{\mathcal{M}}(T)$.

Theorem 3.3.5 (Effective Unions in $\mathbf{Sub}_{\mathcal{M}}(T)$ for $(\mathbf{AGraphs}_{\mathbf{ATG}}, \mathcal{M})$)**.** *Let* T *be an object in* $(\mathbf{AGraphs}_{\mathbf{ATG}}, \mathcal{M})$*, where* $\mathcal{M}$ *contains all monomorphisms, which are isomor-*

phisms on the data part. The union $A \cup B$ of two objects A and B in $\mathbf{Sub}_{\mathcal{M}}(T)$ is effective, i.e. it can be constructed as the pushout (1) over the intersection $A \cap B$ with the induced $\mathcal{M}$-morphism $u : A \cup B \to T$ of the pushout (1).

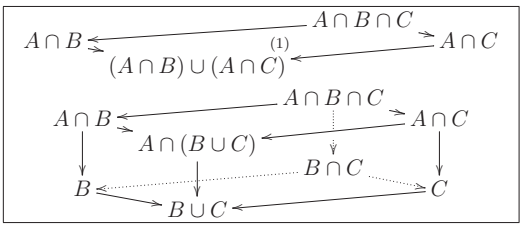


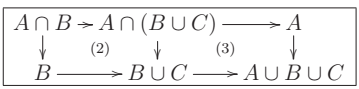
Proof. The intersection exists by Lemma 3.3.3 and the pushout exists, because $p_A \in \mathcal{M}$. The induced morphism u is monomorphic using Thm. 4.7 in [LS04] and the existence of pullbacks along $\mathcal{M}$-morphisms. It remains to show that u is also an $\mathcal{M}$-morphisms. This means that u is additionally an isomorphism on the data part. Since a and i_A are $\mathcal{M}$-morphisms they are isomorphisms on the data part and using the inverse isomorphism of i_A on its data part we have that u is an isomorphism on the data part by the composition of isomorphisms. □

Theorem 3.3.6 (Distributivity)**.** *Let $\mathbf{C}$ be an $\mathcal{M}$-adhesive category with effective unions and T be an object of $\mathbf{C}$, then the union and intersection constructions in $\mathbf{Sub}_{\mathcal{M}}(T)$ are distributive, i.e.*

$$(i): \quad A \cap (B \cup C) = (A \cap B) \cup (A \cap C) \quad \textit{and}$$
$$(ii): \quad A \cup (B \cap C) = (A \cup B) \cap (A \cup C).$$

Proof. Property (i) : The proof is analogous to the one for Cor. 5.2 in [LS05] concerning adhesive categories and we lift it to $\mathcal{M}$-adhesive categories. Let A, B, C be objects in $\mathbf{Sub}_{\mathcal{M}}(T)$, then (1) is pushout in $\mathbf{C}$ since $\mathbf{C}$ has effective unions. The cube is commutative, because all diagrams in $\mathbf{Sub}_{\mathcal{M}}(T)$ commute and $A \cap C \subseteq A \cap (B \cup C)$, because $C \subseteq B \cup C$. The bottom face is a pushout in $\mathbf{C}$ along an $\mathcal{M}$-morphism, because $\mathbf{C}$ has effective unions. The back faces are pullbacks in $\mathbf{C}$ according to Lem. 3.3.3. The front left face of the cube is a pullback by pullback decomposition of the pullback $(2+3)$. For the analogous reason, the front right face of the cube is a pullback. By the VK-property of $\mathcal{M}$-adhesive categories we derive that the top face of the cube is a pushout and by uniqueness of pushouts we deduce property (i) and by duality in lattices we also have property (ii). □





Based on the notion of $\mathcal{M}$-subobjects and the distributivity law for intersection and union we now present subobject transformation systems (STSs) as formal framework for the concurrent semantics of $\mathcal{M}$-adhesive transformation systems. This concept generalises the the notion of elementary nets, which form the category of process nets for P/T Petri nets, in the way that STSs form the category of process transformation systems for $\mathcal{M}$-adhesive transformation systems. As we shall see by Fact 3.3.11, the typical effect occurring in elementary nets – namely the situation of contact – also appears in the setting of STSs and forms an addition application conditions for the transformation rules. Thus, we first introduce the general setting of STSs and thereafter, we show how the process of a transformation sequence in an $\mathcal{M}$-adhesive transformation system is constructed as an STS together with a morphisms into the original $\mathcal{M}$-adhesive transformation system.

Definition 3.3.7 (STS with NACs)**.** *A* Subobject Transformation System with NACs $\mathcal{S} = (T, P, \pi)$ *over an $\mathcal{M}$-adhesive category* $\mathbf{C}$ *with effective unions consists of a super object $T \in \mathbf{C}$, a set of rule names P – also called productions – and a function π, which maps a rule name $p \in P$ to a* rule with NACs $(\langle L_q, K_q, R_q\rangle, N)$*, where L_q, K_q, and R_q are objects in $\mathbf{Sub}_{\mathcal{M}}(T)$, $K_q \subseteq L_q$, $K_q \subseteq R_q$ and $\mathbf{N} = \langle N[1], N[2], \ldots, N[k]\rangle$ is an ordered list of negative application conditions with $L \subseteq N[i] \subseteq T$, where $N[i]$ denotes the i(th) element of N.*

Direct derivations ($G \overset{q}{\Longrightarrow} G'$) with NACs in an STS correspond to transformation steps with NACs in $\mathcal{M}$-adhesive transformation systems, but the construction is simplified, because morphisms between two subobjects are unique. This means that there is no need for pattern matching and the match of the left hand side in G is fixed and does not have to be specfied in ($G \overset{q}{\Longrightarrow} G'$). Thus, there is also no need for performing pattern matching for NACs – there is at most one occurrence of N in the object G. As we shall see later in Fact 3.3.12, given a rule q and a subobject G, then the context subobject D of G w.r.t. q is unique (not only up to isomorphism), if it exists. Thus, direct derivations starting at a subobject G via a rule q in an STS are unique, if they exist.

Definition 3.3.8 (Direct Derivations with NACs in an STS)**.** *Let $\mathcal{S} = \langle S_0, T, P, \pi\rangle$ be a Subobject Transformation System with NACs, $\pi(q) = (\langle L, K, R\rangle, N)$ be a production with NACs, and let G be an object of $\mathbf{Sub}_{\mathcal{M}}(T)$. Then there is a* direct derivation with NACs *from G to G' using q, written $G \overset{q}{\Longrightarrow} G'$, if $G' \in \mathbf{Sub}_{\mathcal{M}}(T)$, for each $N[i]$ in N: $N[i] \not\subseteq G$, and there is an object $D \in \mathbf{Sub}_{\mathcal{M}}(T)$ such that:*

$$(i) \quad L \cup D = G; \qquad (ii) \quad L \cap D = K;$$
$$(iii) \quad D \cup R = G', \text{ and } \quad (iv) \quad D \cap R = K.$$

Given subobjects L and R, considered as left- and right-hand sides of a rule, there is a canonical choice for the interface K, namely $K = L \cap R$. In this case, we say the rule is pure and consequently, an STS is pure if all its rules are pure.

Definition 3.3.9 (Pure STS)**.** *An STS is* pure*, if $K_q = L_q \cap R_q$ holds for all its rules q.*

Intuitively, in a pure STS no rule deletes and produces again the same part of a subobject: this terminology is adapted from the theory of Elementary Net Systems, where a system which does not contain transitions with a self-loop is called "pure".

The next technical lemma will be used several times along the paper: It provides a simple set-theoretical syntax for expressing the fact that a commutative square in $\mathbf{Sub}_{\mathcal{M}}(T)$ is a pushout in $\mathbf{C}$.

Lemma 3.3.10 (Characterization of Unions and Intersections)**.** *Assume that $\mathbf{C}$ is an $\mathcal{M}$-adhesive category with effective unions, that $T \in \mathbf{C}$, and that $(\dagger)$ is a square in $\mathbf{Sub}_{\mathcal{M}}(T)$. In $\mathbf{Sub}_{\mathcal{M}}(T)$, we refer by an object to its equivalence class. Then the following are equivalent:*

(1) $(\dagger)$ *is a pushout in* $\mathbf{C}$
(2) $B \cap C = A$ *and* $D = B \cup C$ *in* $\mathbf{Sub}_{\mathcal{M}}(T)$
(3) $B \cap C \subseteq A$ *and* $D \subseteq B \cup C$ *in* $\mathbf{Sub}_{\mathcal{M}}(T)$.

$$\begin{array}{ccc} A & \rightarrowtail & B \\ \downarrow & (\dagger) & \downarrow \\ C & \rightarrowtail & D \end{array}$$

Proof. $(1 \Rightarrow 2)$ The square $(\dagger)$ is a pushout in $\mathbf{C}$ along an $\mathcal{M}$-morphism, therefore a pullback [EEPT06]. Since $B \cap C$ is also a pullback of $C \to D$ and $B \to D$ (given that there is a morphism $D \to T$), we have $B \cap C = A$ in $\mathbf{Sub}_{\mathcal{M}}(T)$ and thus $D = B \cup C$ in $\mathbf{Sub}_{\mathcal{M}}(T)$.

$(2 \Leftrightarrow 3)$ The identities imply the inclusions. For the other implication, it is sufficient to observe that, given $(\dagger)$, $A \subseteq B \cap C$ and $B \cup C \subseteq D$ hold by the universal properties of $\cap$ and $\cup$, respectively.

$(1 \Leftarrow 2)$ Diagram $(\dagger)$ is a pushout in $\mathbf{C}$, because $\mathbf{C}$ has effectidve unions(see Def. 3.3.2). □

It is instructive to consider the relationship between a direct derivation in an STS and the usual notion of a DPO transformation step in an $\mathcal{M}$-adhesive category. It is possible to make this comparison, since one can consider a rule $L_p \supseteq K_p \subseteq R_p$ as the underlying span of $\mathcal{M}$-morphisms in $\mathbf{C}$.

We shall say that there is a *contact situation* for a rule $\langle L, K, R\rangle$ at an $\mathcal{M}$-subobject $G \supseteq L \in \mathbf{Sub}_{\mathcal{M}}(T)$ if $G \cap R \not\subseteq L$. Intuitively this means that part of the $\mathcal{M}$-subobject G is created but not deleted by the rule: if we were allowed to apply the rule at this match via a DPO transformation step, the resulting object would contain the common part twice and consequently the resulting morphism to T would not be an $\mathcal{M}$-morphism; i.e., the result would not be a $\mathcal{M}$-subobject of T. The next result clarifies the relationship between the definition of direct derivation in $\mathbf{Sub}_{\mathcal{M}}(T)$ and the standard definition used in the DPO approach [EEPT06]: essentially, STS direct derivations and DPO transformation steps coincide if there is no contact.

Fact 3.3.11 (STS Derivations are Contact-Free Double Pushouts)**.** *Let $\mathcal{S} = \langle T, P, \pi \rangle$ be an STS over an $\mathcal{M}$-adhesive category $\mathbf{C}$ with effective unions, $\pi(q) = \langle L, K, R \rangle$ be a rule, and G be an object of $\mathbf{Sub}_{\mathcal{M}}(T)$. Then $G \overset{q}{\Longrightarrow} G'$ iff $L \subseteq G$, $G \cap R \subseteq L$, and there is an object D in $\mathbf{C}$ such that the following diagram consists of two pushouts in $\mathbf{C}$.*

$$\begin{array}{ccccc} L & \xleftarrow{l} & K & \xrightarrow{r} & R \\ {\scriptstyle m}\downarrow & (1) & \downarrow{\scriptstyle k} & (2) & \downarrow{\scriptstyle n} \\ G & \xleftarrow[f]{} & D & \xrightarrow[g]{} & G' \end{array}$$

Proof. ($\Rightarrow$) Suppose that $G \overset{q}{\Longrightarrow} G'$. Then by Definition 3.3.8 there is an object $D \in \mathbf{Sub}_{\mathcal{M}}(T)$ such that (i) $L \cup D = G$ and (ii) $L \cap D = K$, so clearly $L \subseteq G$ and by the conclusion of Lemma 3.3.10, the left square (1) is a pushout in $\mathbf{C}$. Furthermore, (iii) $D \cup R = G'$ and (iv) $D \cap R = K$, and thus (2) is a pushout in $\mathbf{C}$ as well. The fact that $G \cap R \subseteq L$ can be shown as follows, using (i) and (iv):

$$G \cap R \overset{(i)}{=} (L \cup D) \cap R \overset{(*)}{=} (L \cap R) \cup (D \cap R) \overset{(iv)}{=} (L \cap R) \cup K \subseteq L$$

where $(*)$ holds by distributivity of $\mathbf{Sub}_{\mathcal{M}}(T)$.

($\Leftarrow$) Suppose that the squares (1) and (2) are pushouts in $\mathbf{C}$, $L \subseteq G$ (3) and $G \cap R \subseteq L$ (4). Since $G \in \mathbf{Sub}_{\mathcal{M}}(T)$ and (3), all arrows of (1) are in $\mathbf{Sub}_{\mathcal{M}}(T)$ and by Lemma 3.3.10 we have (ii) $L \cap D = K$ and (i) $L \cup D = G$.

Clearly $K \subseteq D \cap R$. Now $D \cap R = D \cap G \cap R \subseteq_{(4)} D \cap L = K$ and thus we conclude that condition (iv) : $K = D \cap R$ holds. Because square (2) is a pushout it follows that $(iii) : D \cup R = G'$. □

The following example shows that in the presence of a contact situation, a double-pushout diagram in $\mathbf{C}$ does not correspond in general to a direct derivation in the STS. More precisely, let $\mathbf{C}$ be the ($\mathcal{M}$-adhesive) category of sets and functions, and let $T = \{\bullet\}$ be a singleton set. Then the top span is a rule in $\mathbf{Sub}_{\mathcal{M}}(T)$, and arrow m is in $\mathbf{Sub}_{\mathcal{M}}(T)$ as well, but condition $G \cap R \subseteq L$ is not satisfied. The double-pushout diagram can be completed in $\mathbf{Sets}$ as shown, but the resulting set G' is not a subobject of T.

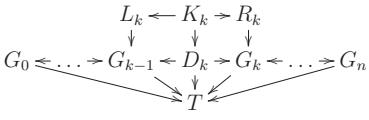
$$\begin{array}{ccccc} L = \emptyset & \xleftarrow{l} & K = \emptyset & \xrightarrow{r} & R = \{\bullet\} \\ {\scriptstyle m}\downarrow & (1) & \downarrow{\scriptstyle k} & (2) & \downarrow{\scriptstyle n} \\ G = \{\bullet\} & \xleftarrow[f]{} & D = \{\bullet\} & \xrightarrow[g]{} & G' = \{\bullet, \bullet\} \end{array}$$

As a consequence of the fact that a direct derivation in an STS implies a transformation step in the standard DPO approach, we can immediately derive its determinacy below.

Fact 3.3.12 (Determinacy of STS derivations)**.** *Suppose that $\mathcal{S} = \langle T, P, \pi \rangle$ is an STS over an $\mathcal{M}$-adhesive category with effective unions, q is a rule, and G is an object of $\mathbf{Sub}_{\mathcal{M}}(T)$. Then the context of G w.r.t. q is unique, if it exists. As a consequence the target of a direct derivation is determined uniquely as well: if $G \Rightarrow^{q} G'$ and $G \Rightarrow^{q} G''$ then $G' = G''$.*

Proof. If D is a context of G w.r.t. q, by Fact 3.3.11 it is a pushout complement of $K \subseteq L$ and $L \subseteq G$ in $\mathbf{C}$. The statement follows from the uniqueness up to isomorphism of pushout complements along $\mathcal{M}$-morphisms in $\mathcal{M}$-adhesive categories [EEPT06] and the fact that morphisms in $\mathbf{Sub}_{\mathcal{M}}(T)$ are always unique. □

Given a transformation sequence d the construction of its subobject transformation system $STS(d)$ is performed as defined below according to [CHS08] and the extensions to $\mathcal{M}$-adhesive categories, general matching and NACs in [HE08, Her09a].
In order to construct the STS for a transformation sequence $d = (d_1; \ldots; d_n)$ we compute the colimit T of the sequence of DPO diagrams, where all morphisms are injective. Thus, all objects and morphisms of this diagram are in the category $\mathbf{Sub}_{\mathcal{M}}(T)$. The NACs of the rules do not occur in this diagram.



For the rest of the paper, we consider only transformation sequences such that the colimit T is finite, i.e. has finitely many $\mathcal{M}$-subobjects. For typed attributed graphs, this means that T is finite on the structural part and the carrier sets of the data algebra for the attribution component may by infinite ($\mathcal{M}$-morphisms in $\mathbf{AGraphs}_{ATG}$ are isomorphisms on the data part). Thus, $\mathbf{Sub}_{\mathcal{M}}(T)$ is a finite lattice. When constructing an STS for a given transformation sequence of an $\mathcal{M}$-adhesive transformation system TS, finiteness is guaranteed if each rule of TS has finite left- and right-hand sides, and if the start object of the transformation sequence is finite, where finite again means that there are finitely many $\mathcal{M}$-subobjects.

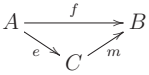
During the generation of an STS with NACs from a given transformation sequence each rule is equipped with a list of NACs, i.e., those obtained as "instances" of the original NACs in the colimit object T.

Definition 3.3.13 (Instantiated NACs)**.** *Let $d = (d_1; \ldots; d_k; \ldots; d_n)$ be a transformation sequence respecting NACs in an $\mathcal{M}$-adhesive transformation system with NACs. Let further $\langle p, \mathbf{N} \rangle$ be the rule with NACs used in transformation step d_k, let T be the colimit object of the transformation sequence, and let $in_T(L_p)$ be the injection in T of the left-hand side of p. Let $n : L_p \hookrightarrow N$ be a NAC of p, then an* instantiated NAC of n in T *is a subobject $[j : N \hookrightarrow T] \in \mathbf{Sub}_{\mathcal{M}}(T)$ such that $j \circ n = in_T(L_p)$. The set of all instantiated NACs in T of all NACs of a rule p is denoted by $NACS_T(p)$.*

Remark 3.3.14. *Note that given a NAC, then the set of its instantiated NACs may be empty, which means that the NAC cannot be found within T. Furthermore, since we require T to be finite, we get a finite list for each NAC.*

For transformation systems with general matching the construction of an STS requires an instantiation of the transformation diagrams as well as presented in [HE08, Her08a, Her08b]. General matching is important for e.g. attribution. Rules may use terms to specify attribute values and for this reason, two terms may be matched to the same value. This implies that the match in this step is not injective and therefore not an $\mathcal{M}$-morphism. For this reason the transformation sequence is instantiated in the way that all identifications are performed to the rules itself yielding new transformation diagrams with matches in $\mathcal{M}$. This instantiation is based on $\mathcal{E}$-$\mathcal{M}$ factorizations of the matches.

Definition 3.3.15 ($\mathcal{E}$-$\mathcal{M}$ factorization)**.** $\mathbf{C}$ *has an $\mathcal{E}$-$\mathcal{M}$ factorization for given morphism classes $\mathcal{E}$ and $\mathcal{M}$ if for each f there is a decomposition, unique up to isomorphism, $f = m \circ e$ with $e \in \mathcal{E}$ and $m \in M$.*



Usually $\mathcal{E}$ is a subclass of epimorphisms and $\mathcal{M}$ is a subclass of monomorphisms. In the context of the $\mathcal{M}$-adhesive category $\mathbf{AGraphs_{ATG}}$ of typed attributed graphs the class $\mathcal{E}$ contains all epimorphisms and $\mathcal{M}$ is the distinguished class of $\mathcal{M}$-adhesive category, i.e. it consists of all monomorphisms that are isomorphisms on the data part. We now show that the category $\mathbf{AGraphs_{ATG}}$ has $\mathcal{E}$-$\mathcal{M}$-factorizations for a restricted class of morphisms, which is enough in our case, because matches of transformation sequences are included in this class.

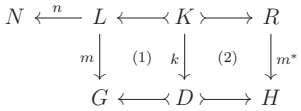
Theorem 3.3.16 ($\mathcal{E}$-$\mathcal{M}$ factorization for $(\mathbf{AGraphs_{ATG}}, \mathcal{M})$)**.** *Let $\mathcal{E}$ be the class of all epimorphisms in the $\mathcal{M}$ adhesive category $\mathbf{AGraphs_{ATG}}$ with $\mathcal{M}$ the class of monomorphism that are isomorphisms on the data part. Let $f = (f_G, f_D) : AG \to AH$ be a morphism in $\mathbf{AGraphs_{ATG}}$, where $AG = (G, A_G)$ and $AH = (H, A_H)$ are attributed graphs, where algebra $A_G = T_\Sigma(X)$ and algebra A_H is term generated. Then, there is an $\mathcal{E}$-$\mathcal{M}$ factorization $(e, m) = ((e_G, e_D), (m_G, m_D))$ for f. The morphisms (e_G, m_G) are given by the epi-mono factorization for typed E-graphs and the morphisms (e_D, m_D) are given by the the epi-mono factorization for $\mathbf{Alg}(\mathbf{\Sigma})$.*

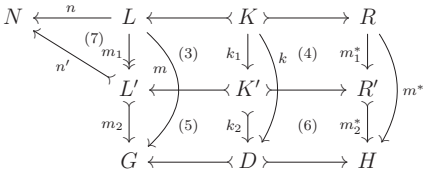
Proof. We can apply the $\mathcal{E}$-$\mathcal{M}$-pair factorization in item 1 of Ex. 9.22 in [EEPT06] using the jointly surjective pair $(f_1, f_2) = (f, f)$ and derive the epimorphism e by $(e_1, e_2) = (e, e)$ and the monomorphism m. It remains to show that $m \in \mathcal{M}$. Since A_H is term generated we know that $f_D = xeval(ass)$ is surjective and by decomposition also m_D is surjective. By m_D being mono according to the factorization we have that m_D is an isomorphism. Thus, $m \in \mathcal{M}$. □

Remark 3.3.17. *In the case that we consider transformation sequences with algebras that are not term generated we can still perform a factorization $AG \xrightarrow{e} AG' \xrightarrow{m} AH$, but this time with e being a general morphism and $m \in \mathcal{M}$. In this case AG' is obtained as before on the graph part, but for the data part we use the A-quotient term algebra extended by a new variable for each element not reached by $xeval(ass) : T_{Sigma}(X) \to A_H$. Based on this construction we can also construct the STS, because we only use the lower part of the factorized transformation diagrams, where only $\mathcal{M}$-morphisms occur.*

Based on the factorization of the matches we instantiate the transformation steps of a transformation sequence leading to a new transformation sequence, where all matches are $\mathcal{M}$-morphisms. Note in particular, that if all matches are $\mathcal{M}$-morphisms already the instantiated transformation sequence coincides with the given original transformation sequence.

Definition 3.3.18 (Instantiation of a Transformation Sequence)**.** *Let $G \overset{p,m}{\Longrightarrow} H$ be a transformation step in an $\mathcal{M}$-adhesive transformation system with $\mathcal{E}$-$\mathcal{M}$ factorization for the morphism class of the matches. The instantiated transformation step $G \overset{p,m_2}{\Longrightarrow} H$ is given by diagrams (5) and (6) below, which are constructed as follows.*





The match m is factorized in the $\mathcal{E}$-morphism m_1 and the $\mathcal{M}$-morphism m_2. Diagram (5) is constructed as pullback, implying that $k_2 \in \mathcal{M}$, because $\mathcal{M}$ is closed under pullbacks. Morphism k_1 is obtained as the induced morphism from the pullback (1) into K' and we have that (3) and (5) are pushouts and pullbacks by the $\mathcal{M}$ pushout-pullback decomposition lemma (item 2 of Thm. 4.26 in [EEPT06]). Diagram (4) is obtained as a pushout and the morphism m_2^* is derived as the induced morphism from the pushout (4) into H. Thus, (6) is a pushout by pushout decomposition making $((5),(6))$ a DPO diagram. For each NAC $n : L \to N$ of p check, if there is an $\mathcal{M}$-morphism $n' : L' \to N$, such that (7) commutes. If it exists, it is unique, because it is part of an $\mathcal{E}$-$\mathcal{M}$ factorization of n.

Let d be a transformation sequence in an $\mathcal{M}$-adhesive transformation system with $\mathcal{E}$-$\mathcal{M}$ factorization for the morphism class of the matches. The instantiated transformation sequence d' is derived by instantiating each transformation step as defined above.

Example 3.3.19 (Instantiation)**.** *We show the instantiation of the transformation step d in Fig. 3.10. In order to illustrate the effects of non-injective matches we use a simple attributed graph transformation system with type graph TG and a rule "setValue" given by the upper line of he transformation diagram. The attribute variables $x1$ and $x2$ in L*

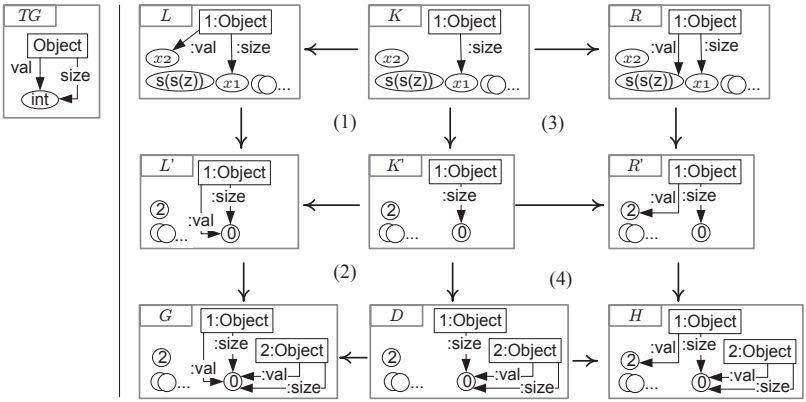


Figure 3.10: Instantiated transformation step d with rule setValue, type graph TG

are identified by the match to the same value 0 *in* G*. Squares* $(1+2)$ *and* $(3+4)$ *define the transformation step* d *and squares* (2) *and* (4) *specify the instantiated transformation step. All squares* (1) *through* (4) *are pushouts. Note in particular that in pushout* (1) *the evaluation of* $x2$ *in graph* L' *is induced by the evaluation in graph* K'*, because* L *and* K *have the same term algebra with variables and* L', K' *have the same data algebra.*

Note that for the category $\mathbf{AGraphs}_{ATG}$ we have that the left hand side of the instantiated rule p' is given by $L' = m(L)$.

Using the instantiation of a transformation sequence we will apply the process construction STS below on the instantiated transformation diagram with $\mathcal{M}$-morphisms. Thus, the results for the analysis of permutation equivalence based on the derived STS in Secs. 3.4 and 3.5 are ensured for the lower DPO diagrams of an instantiated transformation sequence d. But the computed equivalent transformation sequences for the lower part lead to equivalent transformation sequences of d by composing the new DPO diagrams with the corresponding but unchanged DPO diagrams of the upper line of the instantiation. Furthermore, we show by Lem. 3.3.20 below that NAC consistency for the original rule is equivalent to NAC consistency of the instantiated rule. Note that permutation equivalence requires that matches are equivalent according to switch equivalence without NACs based on the Local Church Rosser Thm. This implies that given an $\mathcal{E}$-$\mathcal{M}$ factorization $m_2 \circ e_1$ of a match m we have that each equivalent match m_3 can be decomposed to $m_3 = m'_3 \circ e_1$.

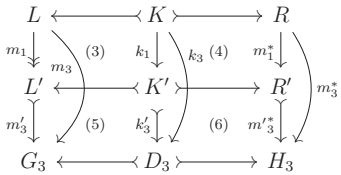
Lemma 3.3.20 (Soundness and Completeness of Rule Instantiation)**.** *Given a transformation step* $G \overset{p,m}{\Longrightarrow} H$ *and the instantiated step* $G \overset{p',m'}{\Longrightarrow} H$ *acc. to Def. 3.3.18 and given a match* $m'_3 : L' \to G_3$ *with* $m'_3 \in \mathcal{M}$*. Then:*

there is a transformation step $G_3 \overset{p',m'_3}{\Longrightarrow} H_3$ *via* p'

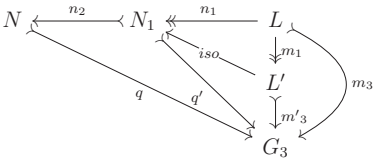
iff

there is a transformation step $G_3 \overset{p,m_3}{\Longrightarrow} H_3$ *via* p *with* $m_3 = m'_3 \circ m_1$.

Proof. Without considering the NACs we have that the transformation step via p' can be composed with the diagrams (3) and (4) acc. to Def. 3.3.18 leading to a transformation step via p and match m_3. Vice versa, for a transformation step via p and match m_3 we can conclude that K' is isomorphic to the pullback of $(L' \to G_3 \leftarrow D_3)$ using the $\mathcal{M}$ pushout-pullback lemma (item 2 of Thm. 4.26 in [EEPT06]) and uniqueness of pushout complements for rules in $\mathcal{M}$-adhesive transformation systems and we derive pushouts (3) and (5). The comatch m'^*_3 of the instantiated rule is induced by pushout (4). Finally, (6) is a pushout by pushout decomposition.



We now consider the NACs. For the transformation diagram with step $G_3 \overset{p',m'_3}{\Longrightarrow} H_3$ we have that a NAC occurrence $q' : N' \to G_3$ of the instantiated rule p' induces a NAC occurrence $q : N \to G_3$ of the original rule p, because (7) in Def. 3.3.18 commutes. We now show that also a NAC occurrence $q : N \to G_3$ of p induces a NAC occurrence $q' : N' \to G_3$ of p'. Consider a NAC occurrence $q : N \to G_3$ of p via match $m_3 = m'_3 \circ m_1$, i.e. $q \circ n = m'_3 \circ m_1$ with $q \in \mathcal{M}$ as shown below.



We can perform an $\mathcal{E}$-$\mathcal{M}$ factorization of n via N_1 and have the morphsim $q' = q \circ n_2$. Now by uniqueness of $\mathcal{E}$-$\mathcal{M}$ factorizations we have that $L' \cong N_1$ via $iso : L' \to N_1$ and thus, $n' = n_2 \circ iso : L' \to N$ is a NAC of p' and thus, q is an occurrence of the NAC n' of p' in G_3. □

Based on the instantiation of transformation sequences and the instantiation of NACs we now define the construction of an STS $STS(d)$ for a given transformation sequence d.

Definition 3.3.21 (STS-compatible $\mathcal{M}$-adhesive Transformation System)**.** *An $\mathcal{M}$-adhesive transformation system* TS *over an $\mathcal{M}$-adhesive category* **C** *is STS-compatible, if* **C** *has effective unions and* **C** *has $\mathcal{E}$-$\mathcal{M}$-factorization for matches in* TS.

Definition 3.3.22 (STS of a Transformation Sequence with NACs)**.** *Let $d = (G_0 \overset{q_1,m_1}{\Longrightarrow} \ldots \overset{q_n,m_n}{\Longrightarrow} G_n)$ be a transformation sequence respecting NACs in an STS-compatible $\mathcal{M}$-adhesive transformation system. The* STS with NACs generated by d *is given by $STS(d) = (T, P, \pi)$ and its components are constructed as follows. Let d' be the instantiated transformation sequence of d according to Def. 3.3.18, then $STS(d) = (T, P, \pi)$, where T is the colimit of the DPO-diagrams given by d', $P = \{i \mid i \in [n]\}$ is a set that contains a rule occurrence name for each rule occurrence in d', and the mapping π is given as follows: $\pi(k) = (\langle L_k \supseteq K_k \subseteq R_k \rangle, N_k)$ using the embeddings into T, where N_k is an ordered list of the instantiated NACs given by the set $Nacs_T(q_k, m_k)$ acc. to Def. 3.3.13.*

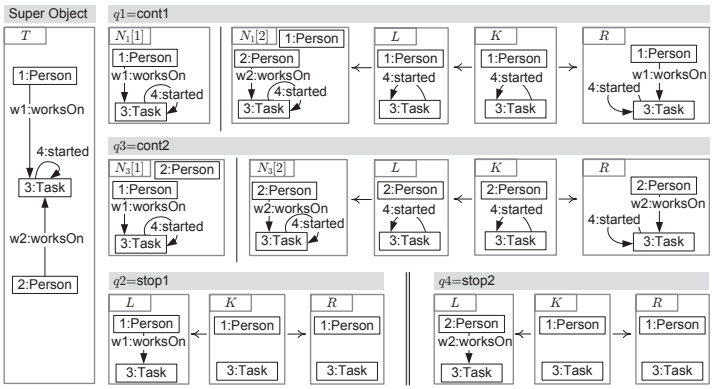


Figure 3.11: Derived Subobject Transformation System $STS(d)$

Example 3.3.23 (Derived STS $STS(d)$)**.** *For the transformation sequence d in Ex. 3.2.1 we derive the STS as shown in Fig. 3.11. The super object T is derived by taking the first graph of the transformation sequence and adding the items, which are created during the transformation, i.e. the two edges of type "worksOn". The transformation sequence d involves the rules "continueTask" and "stopTask" and thus, the derived STS contains the rule name $P = \{1, 2, 3, 4\}$, where the NACs of the rule "continueTask" are instantiated. For an improved intuition, we denote rule 1 by "cont1", 2 by "stop1", 3 by "cont2", and 4 by "stop2" to indicate the name of the original rule and the amount of occurrences of this rule till the current transformation step.*

By Def. 3.3.24 below, we relate sequences of rule names and corresponding derivations in an STS. The mapping drv shows that we can uniquely specify derivations by their sequence of rule names.

Definition 3.3.24 (STS-Derivation of an STS-sequence)**.** *Given an STS $\mathcal{S}$ and a subobject G_0 in $\mathcal{S}$. The partial mapping $drv : R^* \to DRV(\mathcal{S})$ maps a sequence $s = \langle q_1; \ldots; q_n \rangle$ to a derivation $d_S = (G_0 \overset{q_1}{\Longrightarrow} G_1 \Rightarrow \ldots \overset{q_n}{\Longrightarrow} G_n)$ in $\mathcal{S}$, if the derivation d_S exists.*

Based on the construction of the STS of a derivation according to Def. 3.3.22 we now define the interleaving semantics of $\mathcal{M}$-adhesive transformation systems in a similar way as an occurrence grammar in the setting of adhesive transformation systems in [BCH+06]. Each transformation sequence d is mapped to its corresponding STS $STS(d)$ together with a relating mapping v from STS to the transformation system. At the end of Sec. 3.4 we then show by Thm. 3.4.12 that this construction specifies all permutation-equivalent transformation sequences of d. This means that each transformation sequence is related to a process model that specifies all its equivalent interleavings. In a future step we will extend the interleaving analysis to a true concurrent analysis, i.e. including the merging of independet steps to concurrent steps.

Definition 3.3.25 (Process of an $\mathcal{M}$-adhesive Transformation Sequence with NACs)**.** *Let $d = (G_0 \overset{q_1,m_1}{\Longrightarrow} \ldots \overset{q_n,m_n}{\Longrightarrow} G_n)$ be a transformation sequence respecting NACs in an STS-compatible $\mathcal{M}$-adhesive transformation system* TS $= (TG, P_{\text{TS}}, \pi_{\text{TS}})$*. The* process *$Prc(d) = (STS(d), v)$ of d consists of the derived STS $STS(d) = (T, P, \pi)$ of d together with the mapping $v : STS(d) \to$ TS given by $v = (v_T, v_P, v_\pi)$ with $v_T = type_T : T \to TG$, $v_P(i) = q_i$ for each step i of d and $v_\pi : P \to TRAFO(\mathbf{C}, \mathcal{M})$, where v_π maps each rule name i in $STS(d)$ to the upper DPO diagram of the instantiation of the transformation step $(G_{i-1} \overset{q_i,m_i}{\Longrightarrow} G_i)$ according to Def. 3.3.18.*

According to Fact 3.3.11 a direct derivation in an STS induces a DPO diagram in the underlying $\mathcal{M}$-adhesive category $(\mathbf{C}, \mathcal{M})$. Therefore, a derivation in an STS, specified by its sequence of rule names, gives rise to a transformation sequence in $(\mathbf{C}, \mathcal{M})$. Moreover, if the STS is part of the process of a transformation sequence d in an STS-compatible $\mathcal{M}$-adhesive transformation system TS, then we can additionally compose the derived DPO diagrams by Fact 3.3.11 with the upper instantiation diagrams of d constructed for $Prc(d)$ and derive a transformation sequence in TS. In order to make this relation precise in Def. 3.3.26 below, we denote by $TRAFO(\mathbf{C}, \mathcal{M})$ the set of all transformation sequences in the $\mathcal{M}$-adhesive category $(\mathbf{C}, \mathcal{M})$ and by $DRV(\mathcal{S})$ the set of all STS derivations in the STS $\mathcal{S}$.

Definition 3.3.26 (Transformation Sequence of an STS-sequence)**.** *Let $d_S = (G_0 \overset{q_1}{\Longrightarrow} G_1 \Rightarrow \ldots \overset{q_n}{\Longrightarrow} G_n)$ be a derivation in the STS $\mathcal{S}$ of a process $Prc(d) = (\mathcal{S}, v_\pi)$ of a derivation d in an STS-compatible $\mathcal{M}$-adhesive transformation system over the category* $\mathbf{C}$

and let $s = \langle q_1; \ldots; q_n \rangle$ denote the sequence of the rule occurrences according to d_S. Then, the partial mapping $trafo : R^* \to TRAFO(\mathbf{C}, \mathcal{M})$ maps the sequence s to the sequence of DPO diagrams (transformation steps) in $\mathbf{C}$ for each derivation step $G_{i-1} \overset{q_i}{\Longrightarrow} G_i$ in $\mathcal{S}$, where each step i in TS *is given by the composition of the DPO diagram $v_\pi(i)$ and the the DPO-diagram of $G_{i-1} \overset{q_i}{\Longrightarrow} G_i$ according to Fact 3.3.11.*

Moreover, the partial mapping $seq : TRAFO(\mathbf{C}, \mathcal{M}) \to R^*$ maps the transformation sequence $trafo(s)$ to the sequence s of rule names in $\mathcal{S}$.

By Fact 3.3.27 below we show that the partial mappings seq and $trafo$ are well defined and are inverse to each other.

Fact 3.3.27 (Correspondence between Transformation and Rule Sequences)**.** *The partial mappings $trafo$ and seq are well defined, i.e. they are partial functions. Moreover, let $d_S = (G_0 \overset{q_1}{\Longrightarrow} G_1 \Rightarrow \ldots \overset{q_n}{\Longrightarrow} G_n)$ be a derivation in a process $Prc(d)$ of a transformation sequence, then $d = trafo(seq(d))$ and $s = seq(trafo(s))$.*

Proof. By Fact 3.3.11 we know that derivations in an STS induce DPO diagrams and by Lem. 3.3.20 we know that they can be composed with the upper DPO diagrams of the instantiation. The equations hold, because of the above results and each step i in d is specified by a distinguished rule $q_i = i$ in $Prc(d)$, i.e. $trafo$ and seq are left unique relations. □

In a future step we will generalise the notion of deterministic processes [Bal00, BCH+06] to the case of $\mathcal{M}$-adhesive transformation systems in order provide conditions which characterise the set of processes of transformation sequences.

3.4 Analysis of Permutation Equivalence Based on Subobject Transformation Systems

Based on the process construction of a transformation sequence with NACs given by an STS in Sec. 3.3 this section presents efficient techniques for the analysis of switch equivalence and permutation equivalence of a transformation sequence according to [CHS08, Her09a, Her09b]. The main advantage of first constructing and then analysing the process model instead of directly checking the conditions of equivalence on the transformation sequence itself is that we massively reduce the efforts for pattern matching. The construction of the STS and the dependency relations used for the analysis is shown to be performed in polynomial time (see Thm. 3.4.15) and the reasoning does not involve any further pattern matching but only simple boolean checks instead. Furthermore, we show by Thm. 3.4.12 that the analysis is sound and complete, i.e. given a transformation sequence then the derived set of permutation-equivalent transformation sequences using the constructed STS coincides with the set of permutation-equivalent transformation sequences of

the given one according to the definition of permutation equivalence (Def. 3.2.6). Therefore, the notion of interleaving semantics via processes $Prc(d) = (STS(d), v)$ specifies exactly all possible equivalent interleavings of a transformation sequence d.

Thus, we provide a static analysis for efficiently generating all permutation-equivalent transformation sequences to a given one. Furthermore, equivalent sequences can be compared to find a suitable canonical one. Moreover, the reasoning on equivalence can be performed iteratively, i.e. a transformation sequence can be analysed, thereafter extended by some steps and the previous analysis results can be reused for the corresponding extended STS for the extended transformation sequence. Finally, given a transformation sequence we can check whether a particular step can be shifted a certain amount of steps by checking the conditions of a legal sequence according to Def. 3.4.9.

In [BCH+06] the notion of occurrence grammars defining the process of a transformation sequence is based on compound relations on STS rules, which can be obtained from the first three basic relations in the following table in Def. 3.4.1 as presented in [CHS08]. The next three relations concern conflicts and complete independence. Finally, the two last relations handle dependencies causes by NACs and they were introduced in [Her09a].

Definition 3.4.1 (Relations on Rules)**.** *Let q_1 and q_2 be two rules in an STS with NACs $\mathcal{S} = (T, P, \pi_N)$ with $\pi_N(q_j) = (\langle L_j, K_j, R_j \rangle, \mathbf{N}_j)$ for $j \in \{1, 2\}$ and $\mathbf{N}_j = (N_j[i])_{i=1..n_j}$. The relations on rules are defined on P as follows:*

Name	*Notation*	*Condition*
Read Causality	$q_1 <_{rc} q_2$	$R_1 \cap K_2 \not\subseteq K_1$
Write Causality	$q_1 <_{wc} q_2$	$R_1 \cap L_2 \not\subseteq K_1 \cup K_2$
Deactivation	$q_1 <_{d} q_2$	$K_1 \cap L_2 \not\subseteq K_2$
Forward Conflict	$q_1 \curlywedge q_2$	$L_1 \cap L_2 \not\subseteq K_1 \cup K_2$
Backward Conflict	$q_1 \curlyvee q_2$	$R_1 \cap R_2 \not\subseteq K_1 \cup K_2$
Independence	$q_1 \diamond q_2$	$(L_1 \cup R_1) \cap (L_2 \cup R_2) \subseteq K_1 \cap K_2$
Weak NAC Enabling	$q_1 <_{wen[i]} q_2$	$1 \leq i \leq \lvert\mathbf{N}_2\rvert \ \wedge\ L_1 \cap N_2[i] \not\subseteq K_1 \cup L_2$
Weak NAC Disabling	$q_1 <_{wdn[i]} q_2$	$1 \leq i \leq \lvert\mathbf{N}_1\rvert \ \wedge\ N_1[i] \cap R_2 \not\subseteq L_1 \cup K_2$

Read causality specifies that rule q_1 produces an item that is read by q_2, but not deleted by q_2 and in the case of write causality we have that q_2 also deletes such an item. Deactivation occurs when rule q_2 deletes an item that is read by q_1, but not created. Two rules are in forward conflict if there is an item that is deleted by both of them and they are in backward conflict if there is an item that is produced by both of them. Two rules are independent if they overlap only on items that are neither produced nor deleted by one of the rules. Therefore, the notions of parallel, sequential and co-parallel independence are special cases

of independence, i.e. they are implied by the condition for independence, where parallel independence is given by $(L_1 \cap L_2 \subseteq K_1 \cap K_2)$, sequential independence by $(R_1 \cap L_2 \subseteq K_1 \cap K_2$ and $L_1 \cap R_2 \subseteq K_1 \cap K_2)$, and co-parallel independence is given by $(R_1 \cap R_2 \subseteq K_1 \cap K_2)$.

Rule q_1 weakly enables the rule q_2 at i if q_1 deletes a forbidden part q_2, i.e. an item of the i-th NAC of q_2 that is not contained in L_2. The rule q_2 weakly disables q_1 at i if q_2 produces a piece of the i-th NAC of q_1. It is worth stressing that the relations introduced above are not transitive in general.

Example 3.4.2 (Relations on Rules)**.** *The rules of $STS(d)$ in Fig. 3.11 are related by the following dependencies. For write causality we have "cont1 $<_{wc}$ stop1" and "cont2 $<_{wc}$ stop2". Weak enabling/disabling are shown in the table below, while read causality and deactivation are empty.*

Weak Enabling		*Weak Disabling*	
$stop1 <_{wen[1]} cont1$	$stop2 <_{wen[2]} cont1$	$cont1 <_{wdn[1]} cont1$	$cont2 <_{wdn[2]} cont2$
$stop1 <_{wen[1]} cont2$	$stop2 <_{wen[2]} cont2$	$cont2 <_{wdn[1]} cont1$	$cont1 <_{wdn[2]} cont2$

The next lemma shows that if two productions of an STS are applicable to the same subobject, then in order to check that they are independent it is enough to consider only a subset of the possible dependency relations among them.

Fact 3.4.3 (Characterization of independence in STSs)**.** *Suppose that there are direct derivations $G \stackrel{q_1}{\Longrightarrow} G_1$ and $G \stackrel{q_2}{\Longrightarrow} G_2$ in an STS over an $\mathcal{M}$-adhesive category with effective unions. Then, independence of rules can be characterised as follows:*

1. $q_1 \diamond q_2$
2. $\neg(q_1 \between q_2) \ \wedge \ \neg(q_1 \curlyvee q_2) \ \wedge \ q_1 \not<_d q_2 \ \wedge \ q_2 \not<_d q_1$
3. $L_1 \subseteq D_2 \ \wedge \ L_2 \subseteq D_1 \ \wedge \ \neg(q_1 \curlyvee q_2)$, *where D_1 and D_2 are the contexts of the first and of the second direct derivations, respectively.*

Suppose that there are direct derivations $G \stackrel{q_1}{\Longrightarrow} G_1 \stackrel{q_2}{\Longrightarrow} G_2$. Then, independence of rules can be characterised as follows:

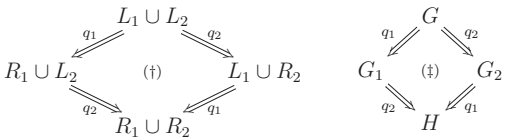
1. $q_1 \diamond q_2$
2. $q_1 \not<_{rc} q_2 \ \wedge \ q_1 \not<_{wc} q_2 \ \wedge \ q_2 \not<_{wc} q_1 \ \wedge \ q_1 \not<_d q_2$
3. $R_1 \subseteq D_2 \ \wedge \ L_2 \subseteq D_1 \ \wedge \ q_2 \not<_{wc} q_1$, *where D_1 and D_2 are the contexts of the first and of the second direct derivations, respectively.*

Proof (Idea). The proof (see Lem. 20 and 21 in [CHS08]) makes use of some laws on the relations using the notion q^{op}, which swaps the LHS and RHS of a rule. Note that in the case that we consider an STS constructed of a derivation the condition $q_2 \not<_{wc} q_1$ is automatically ensured and is only necessary for the general case of an arbitrary STS. □

The next result rephrases in the setting of Subobject Transformation Systems the well-known local Church-Rosser theorem of the DPO approach.

Fact 3.4.4 (Local Church-Rosser for STSs)**.** *Let q_1 and q_2 be two independent productions of an STS $\mathcal{S}$. Then:*

1. *There are direct derivations as in diagram (†) below.*
2. *If there are direct derivations $G \overset{q_1}{\Longrightarrow} G_1$ and $G \overset{q_2}{\Longrightarrow} G_2$, then there is an object H in $\mathbf{Sub}_{\mathcal{M}}(T)$ and direct derivations $G_1 \overset{q_2}{\Longrightarrow} H$ and $G_2 \overset{q_1}{\Longrightarrow} H$, as in diagram (‡) below.*
3. *If there are direct derivations $G \overset{q_1}{\Longrightarrow} G_1 \overset{q_2}{\Longrightarrow} H$, then there is an object G_2 in $\mathbf{Sub}_{\mathcal{M}}(T)$ and direct derivations $G \overset{q_2}{\Longrightarrow} G_2$ and $G_2 \overset{q_1}{\Longrightarrow} H$, as in diagram (‡).*



Proof. (1) It is easy to check that $(L_1 \cup L_2) \overset{q_1}{\Longrightarrow} (R_1 \cup L_2)$ with context $D_1 \overset{\text{def}}{=} L_2 \cup K_1$. In fact, the conditions $(i - iv)$ of Definition 3.3.8 reduce to

1. $L_1 \cup (L_2 \cup K_1) = L_1 \cup L_2$, because $K_1 \subseteq L_1$;
2. $L_1 \cap (L_2 \cup K_1) = (L_1 \cap L_2) \cup (L_1 \cap K_1) = K_1$, by distributivity and independence;
3. $(L_2 \cup K_1) \cup R_1 = R_1 \cup L_2$, because $K_1 \subseteq R_1$;
4. $(L_2 \cup K_1) \cap R_1 = (L_2 \cap R_1) \cup (K_1 \cap R_1) = K_1$, by distributivity and independence.

The other direct derivations are similar, using as contexts (clockwise) $L_1 \cup K_2$, $K_1 \cup R_2$, and $R_1 \cup K_2$.

(2) Let $H \overset{\text{def}}{=} (D_1 \cap D_2) \cup R_1 \cup R_2$, where D_1 and D_2 are the contexts of the first and of the second direct derivations, respectively.

Let us show that $G_1 \overset{q_2}{\Longrightarrow} H$: the proof that $G_2 \overset{q_1}{\Longrightarrow} H$ is analogous. Let $D_2' \overset{\text{def}}{=} (D_1 \cap D_2) \cup R_1$: we show that conditions $(i - iv)$ of Def. 3.3.8 hold for context D_2'. By Fact 3.4.3 we have $(*) : L_2 \subseteq D_1$.

1. $[L_2 \cup D'_2 = G_1]$: We have $G_1 = D_1 \cup R_1$ and $L_2 \cup D'_2 = L_2 \cup ((D_1 \cap D_2) \cup R_1)$ by definition. We show $L_2 \cup (D_1 \cap D_2) = D_1$: $L_2 \cup (D_1 \cap D_2) = (L_2 \cup D_1) \cap (L_2 \cup D_2) =_{(*)} D_1 \cap G = D_1$.

2. $[L_2 \cap D'_2 = K_2]$: Expanding D'_2 and distributing we get $L_2 \cap D'_2 = L_2 \cap ((D_1 \cap D_2) \cup R_1) = (L_2 \cap D_1 \cap D_2) \cup (L_2 \cap R_1)$; the statement follows observing that $(\dagger) L_2 \cap D_1 \cap D_2 = K_2 \cap D_1 =_{(*)} K_2$, and that by independence we have $L_2 \cap R_1 \subseteq K_1 \cap K_2 \subseteq K_2$.

3. $[D'_2 \cup R_2 = H]$: Obvious, by expanding the definitions of H and D'_2.

4. $[D'_2 \cap R_2 = K_2]$: Expanding D'_2 and distributing we get $D'_2 \cap R_2 = (D_1 \cap D_2 \cap R_2) \cup (R_1 \cap R_2)$; analogous to $(\dagger)$, the first argument of the union is K_2, and by independence the second one is included in K_2, allowing us to conclude.

(3) We use the notion $q^{op} = \langle R, K, L \rangle$ for a rule $q = \langle L, K, R \rangle$. This allows us to reduce this point to the previous one by observing that $G \overset{q_1}{\Longrightarrow} G_1$ if and only if $G_1 \overset{q_1{}^{op}}{\Longrightarrow} G$, and $q_1 \diamond q_2$ if and only if $q_1^{op} \diamond q_2$. □

Based on Fact 3.4.4 above we can define switch-equivalence without NACs within STSs, such that the derived switch equivalent sequences define not only transformation sequences in the $\mathcal{M}$-adhesive transformation system, but also within the STS itself. At first we show how the notion of switch equivalence without NACs can be analysed within an STS. On this basis we will show how the additional two relations concerning NACs are used to analyse permutation equivalence for transformation sequences with NACs by refining switch equivalence of transformation sequences without NACs

Definition 3.4.5 (Switch Equivalence of Sequences)**.** *Let $\mathcal{S} = (T, P, \pi)$ be an* STS*, let d be a derivation in $\mathcal{S}$ and let $s = \langle q_1, \ldots, q_n \rangle$ be its corresponding sequence of rule occurrence names. Let q_k,q_{k+1} be independent in $\mathcal{S}$ ($q_k \diamond q_{k+1}$), then the sequence $s' = \langle q_1, \ldots, q_{k+1}, q_k, \ldots, q_n \rangle$ is* switch-equivalent *to the sequence s, written $s \overset{sw}{\sim}_S s'$.* Switch equivalence $\overset{sw}{\approx}_{\mathcal{S}}$ *of sequences is the transitive closure of* $\overset{sw}{\sim}_S$ *.*

By Fact. 3.4.7 below we show that the notion of switch equivalence in an STS is sound and complete with respect to switch equivalence without NAs in the corresponding $\mathcal{M}$-adhesive transformation system. Soundness means that given a transformation sequence d and its derived STS $\mathcal{S}$, then all sequences of rule names s' which are switch-equivalent to the sequence of rule names $seq(d)$ of d within $\mathcal{S}$ actually specify switch equivalent transformation sequences $trafo(s')$ of d in the $\mathcal{M}$-adhesive transformation system. Vice versa, completeness means that for all switch-equivalent transformation sequences d' of d there is a corresponding sequence of rule names $s' = seq(d')$ in $\mathcal{S}$ being switch-equivalent to $seq(d)$ in $\mathcal{S}$.

Remark 3.4.6 (Transformation Sequence and Sequence of Rule Names)**.** *Recall that we use the notions $seq(d), drv(s)$ and $trafo(s)$ according to Def. 3.3.24 and Def. 3.3.26. If a sequence of rule names specifies a derivation in an STS then this derivation is denoted by $drv(s)$. The sequence of rule names of a transformation sequence d in its derived STS is given by $s = seq(d)$ and the transformation sequence of s in the $\mathcal{M}$-adhesive transformation system is given by $trafo(s)$, which is constructed by taking the DPO diagrams induced by $drv(s)$ and composing them with the DPO diagrams of the instantiation of d for the construction of $\mathcal{S}$.*

Fact 3.4.7 (Characterisation of Switch Equivalence)**.** *Let d be a transformation sequence without NACs in an $\mathcal{M}$-adhesive category with effective unions and $\mathcal{S} = STS(d)$, then*

$$seq(d) \overset{sw}{\approx}_{\mathcal{S}} s' \;\Rightarrow\; d \overset{sw}{\approx} trafo(s') \;\wedge\; drv(s') \text{ is a derivation in } STS(d) \text{ and}$$
$$d \overset{sw}{\approx} d' \;\Rightarrow\; seq(d) \overset{sw}{\approx}_{\mathcal{S}} seq(d').$$

Proof (Idea). The proof (see Cor. 1 in [Her09b]) uses the result that independence of two rules involved in two subsequent derivation steps coincides with sequential independence of the corresponding neighbouring transformation steps in the $\mathcal{M}$-adhesive transformation system. The additional result that $drv(s')$ is a derivation in $STS(d)$ follows by stepwise application of Fact 3.4.4 for each switching. □

Remark 3.4.8 (Completeness of switch equivalence)**.** *Switch equivalence without NACs leads to the complete set of equivalent permutations (see Def. 21 and Thm. 25 in [BCH$^+$06] where permutations are called linearisations). This means that the notion of switch equivalence does not restrict the possible equivalent reorderings. In [BCH$^+$06] this result is shown for the setting of adhesive transformation systems and there is no indication that the result shall not hold in the general case of $\mathcal{M}$-adhesive transformation systems.*

The following notion of legal sequences allows us to first characterise permutation-equivalent derivations with NACs within an STS. Afterwards we show that this characterisation is sound and complete for analysing permutation equivalence of transformation sequences with NACs in the original $\mathcal{M}$-adhesive category.

Note that a constructed STS $STS(d)$ is an unfolding of the original transformation sequence. Thus, e.g. for the category **Graphs** we have that elements can be created and deleted, but never re-created after they have been deleted in a derivation of the derived STS, because the colimit construction distinguishes each creation of an element. This implies that each item is either produced by exactly one rule or it is present in the start object and not produced by any rule. Thus, a NAC is satisfied, if an item of the elements it forbids has already been deleted (weak enabling) or such an item is created later (weak disabling). This condition is formalized by the following notion of legal sequences based on the new dependencies. It allows us to first characterise equivalent derivations with NACs within an STS. Afterwards we show that this characterisation is sound and complete for analysing

the permutation equivalence of transformation sequences with NACs in the original $\mathcal{M}$-adhesive category.

Definition 3.4.9 (Legal Sequence)**.** *Let $d = (d_1; \ldots; d_n)$ be a transformation sequence with NACs in an $\mathcal{M}$-adhesive category with effective unions and let $STS(d) = \mathcal{S} = (T, P, \pi)$ be its derived STS with NACs. A sequence $s = \langle\, q_1; \ldots; q_n \,\rangle$ of rule names of P is locally legal at position $k \in \{1, \ldots, n\}$ with respect to d, if each rule name in P occurs exactly once in s and the following conditions hold:*

1. $s \overset{sw}{\approx}_{\mathcal{S}} seq(d)$
2. $\forall$ *NACs* $N_k[i]$ *of* q_k : $\left(\begin{array}{r} \exists\ e \in \{1, \ldots, k-1\} : \quad q_e <_{wen[i]} q_k \text{ or} \\ \exists\ d' \in \{k, \ldots, n\} : \quad q_k <_{wdn[i]} q_{d'}. \end{array} \right)$

The sequence s of rule names is legal with respect to d, if it is locally legal at all positions $k \in \{1, ..., n\}$ with respect to d.

The second condition of Def. 3.4.9 considers NACs and ensures that each NAC of a rule cannot be found in the subobject to which the rule is applied, which is a consequence of Thm. 3.4.12 and explained above. This subsumes the special case of $s = seq(d)$, where we have that $seq(d)$ is always legal with respect to d. The following definition of permutation equivalence of sequences is based on the notion of legal sequences and therefore, it suffices to evaluate the presented relations on rule occurrence names in order to analyse permutation equivalence of sequences.

Definition 3.4.10 (Permutation Equivalence of Rule Sequences)**.** *Let d be a transformation sequence with NACs in an $\mathcal{M}$-adhesive transformation system with effective unions and let $STS(d) = \mathcal{S}$ be its derived subobject transformation system with NACs. Two sequences s, s' of rule names in $\mathcal{S}$ are permutation-equivalent, written $s \overset{\pi}{\approx}_{\mathcal{S}} s'$, if they are legal sequences with respect to d, i.e. they are switch equivalent and for each step and NAC there is an enabling step before or a disabling step thereafter.*

In order to analyse permutation equivalence within STSs we first state by Fact 3.4.11 below that the notion of permutation equivalence of sequences induces derivations within the constructed STS $STS(d)$ of a given transformation sequence d.

Fact 3.4.11 (Permutation-Equivalent Derivations in STSs)**.** *Let d be a transformation sequence with NACs in an $\mathcal{M}$-adhesive transformation system with effective unions and let $STS(d) = \mathcal{S}$ be the derived STS. Then, each sequence s' in $\mathcal{S}$ that is permutation-equivalent to the sequence $seq(d)$ of rule occurrences in d defines a derivation $drv(s')$ in $\mathcal{S}$, i.e. $(s' \overset{\pi}{\approx}_{\mathcal{S}} seq(d)) \Rightarrow drv(s')$ is a derivation in $\mathcal{S}$.*

Proof. The fact is based on Fact 3.4.7 for the case without NACs. The second part of the proof for Lem. 5 in [Her09b] shows that the derivation $drv(s)$ is also NAC-consistent within $\mathcal{S}$. □

Now, we are able to state by Thm. 3.4.12 below that the analysis of permutation equivalence within the framework of STSs is sound and complete. This way, we provide a correct and complete analysis technique for the analysis of interleaving semantics of $\mathcal{M}$-adhesive transformation systems.

Theorem 3.4.12 (Analysis of Permutation Equivalence of Transformation Sequences based on STSs)**.** *Let d be a transformation sequence with NACs in an STS-compatible $\mathcal{M}$-adhesive transformation system and let $Prc(d) = (\mathcal{S}, v)$ be the process of d. Then, the analysis of permutation-equivalence within $\mathcal{S}$ is sound and complete, i.e. each permutation-equivalent rule sequence s' of $seq(d)$ in $\mathcal{S}$ defines a permutation-equivalent transformation sequence $trafo(s')$ of d and vice versa, each permutation-equivalent transformation sequence d' of d defines a permutation-equivalent rule sequence $seq(d')$ of $seq(d)$ in $\mathcal{S}$.*

$$\forall\, s' \in P^* : \quad s' \overset{\pi}{\approx}_{\mathcal{S}} seq(d) \quad \Rightarrow \quad trafo(s') \overset{\pi}{\approx} d \qquad (1)$$

$$\forall\, d' \in TRAFO(\mathbf{C}, \mathcal{M}) : \quad d' \overset{\pi}{\approx} d \quad \Rightarrow \quad seq(d') \overset{\pi}{\approx}_{\mathcal{S}} seq(d) \qquad (2)$$

Proof. The proof (see Thms. 3 and 4 in [Her09b]) uses Facts 3.4.7 and 3.4.11 and shows that the NAC instantiation (Def. 3.3.13) ensures NAC consistency and does not eliminate valid matches. The mapping v explicitly specifies how the typing and the rules of the single transformation steps are related. In particular, the matches of an STS derivation step are extended to matches of the instantiated transformation rule using the instantiation diagrams according to v_π as defined in Def. 3.3.26. □

Based on Thm. 3.4.12 we define for a given transformation sequence d the set $EQU(d)$ of all canonical transformation sequences, which are derived from permutation-equivalent sequences of $seq(d)$ in $STS(d)$. Note that the set $EQU(d)$ is finite, if d is finite.

Definition 3.4.13 (Canonical Equivalent Transformation Sequences)**.** *Let d be a transformation sequence with NACs in an STS-compatible $\mathcal{M}$-adhesive transformation system and let $Prc(d) = (\mathcal{S}, v)$ be the process of d. The set $EQU(d) = \{trafo(s') \mid s' \overset{\pi}{\approx}_{\mathcal{S}} seq(d)\}$ is called* set of canonical permutation-equivalent transformation sequences of d.

Now we can state the second main result of this section by Cor. 3.4.14, which shows that for each transformation sequence d', which is permutation-equivalent to d, there is an isomorphic representative in the set of canonical equivalent transformation sequences $EQU(d)$.

Corollary 3.4.14 (Generation of all Permutation-Equivalent Transformation Sequences based on STSs)**.** *Let d be a transformation sequence with NACs in an $\mathcal{M}$-adhesive category with effective unions and let $STS(d) = \mathcal{S}$ be the derived STS. Then,*

$$\forall\, d' \text{ with } d' \overset{\pi}{\approx} d : \exists\, d'' \in EQU(d) : d' \cong d''.$$

Proof. This is a consequence of Def. 3.4.13 and Thm. 3.4.12, where the transformation sequence d'' is obtained by $trafo(s')$ with s' being the sequence of rule occurrence names that correspond to the steps of d'. □

Furthermore, the construction of the process model of a transformation sequence is efficient as stated by Fact 3.4.15 below. This ensures that the effort for the construction of the presented framework does not lead to efficiency problems of the overall analysis, because the upper bound is given by a polynomial term.

Fact 3.4.15 (Efficient Construction of the Process Model)**.** *Let d be a transformation sequence with NACs in a graph transformation system and let $STS(d) = \mathcal{S}$. If additionally the size of each NAC is bounded by the size of the left hand side of the corresponding rule plus an arbitrary but fixed c, then the complexity of the construction of the process model $\mathcal{S}$ and its dependency relations is in $\mathsf{O}(n^{c+4})$, where n is the length of the input $I = (GG, d)$.*

Proof. If matches are non-injective, the instantiation of the DPO steps leads to smaller objects and reduces the efforts. Thus, for the worst case, we assume that the matches are injective. The super object T is constructed as colimit of d by incremental pushouts for each transformation step and the intermediate colimit object is extended by at most n elements at each step. Thus, this construction has a complexity of $\mathsf{O}(n^2)$. The size of T is at most n, because - in the worst case - T is given as disjoint union of the graphs in d. Furthermore, the construction of S_0 with its embedding to T and the construction of P is linear.

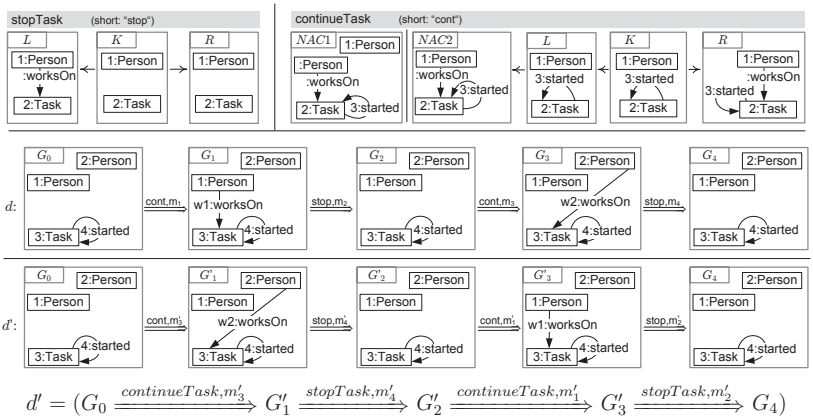
The mapping π is given by composing the morphisms in d with the embeddings to T and instantiation of the NACs. For each transformation step we have at most n NACs of the current rule p. The left hand side of p is already embedded into T and it remains to perform pattern matching for the additional elements (at most c) for each NAC. We derive complexity $\mathsf{O}(n^c \cdot n \cdot n) = \mathsf{O}(n^{2+c})$ and there are at most n^{1+c} NAC-instances for each rule occurrence.

The relations: For each pair of rules we store a Boolean value specifying whether the relation $\diamond$ holds. We have at most n^2 pairs and use the embeddings to T to check whether they overlap only on interface elements of the rules. Thus, we have a complexity of $\mathsf{O}(n^3)$ and a Boolean array of size at most n^2. For each instantiated NAC (at most n^{1+c}) of a rule occurrence q we check the relation $<_{wdn[i]}$ against each other rule q', i.e. whether the rules overlap on L_q and $K_{q'}$ only. We derive the complexity $\mathsf{O}(n^{c+1} \cdot n^2 \cdot n) = \mathsf{O}(n^{c+4})$ and a Boolean array of size at most n^{c+3}. The same procedure is applied for $<_{wen[i]}$ Summing up all steps, we have complexity $\mathsf{O}(n^{c+4})$. □

Remark 3.4.16 (Check for Permutation Equivalence)**.** *If we are interested whether two given transformation sequences d and d' with NACs are permutation-equivalent we can transfer this problem to the usual analysis of switch equivalence without NACs. The reason*

is that we already know that d and d' respect all NACs. But note that this check also involves isomorphism checks, because the modified structure of an intermediate object in d is not related to some structure in d'.

The main advantage of the analysis using STSs is that we do not need to update the graphs of the transformation sequence and we do not need to perform pattern matching for the NACs after each switching. Permutation equivalence of sequences within an STS does only concern the introduced relations on rule occurrence names. This means that we only have to check whether rule components overlap in the specified way. Furthermore, we can compute the relations once and for all and store the results in Boolean arrays. This means that dependencies of steps can be efficiently checked based on the dependency relations.



$$d' = (G_0 \xRightarrow{continueTask,m'_3} G'_1 \xRightarrow{stopTask,m'_4} G'_2 \xRightarrow{continueTask,m'_1} G'_3 \xRightarrow{stopTask,m'_2} G_4)$$

write causality	weak enabling	weak disabling
cont1 $<_{wc}$ stop1	stop1$<_{wen[1]}$cont1	cont1$<_{wdn[1]}$cont1
cont2 $<_{wc}$ stop2	stop1$<_{wen[1]}$cont2	cont1$<_{wdn[1]}$cont2
	stop2$<_{wen[2]}$cont1	cont2$<_{wdn[2]}$cont2
	stop2$<_{wen[2]}$cont2	cont2$<_{wdn[2]}$cont1

Figure 3.12: Permutation-equivalent transformation sequences d, d' and the dependencies in $STS(d)$

Example 3.4.17 (Equivalent Sequence)**.** *The two transformation sequences d and d' in Fig. 3.12 are permutation-equivalent as motivated already in Sec. 3.2. The last two steps of d are moved to the beginning in transformation sequence d'. Using the relations in an STS we derive the dependencies as listed in Fig. 3.12 for the rule occurrences "cont1, cont2, stop1, stop2" in Fig. 3.11. Note that the rule names in a derived STS of a transformation sequence*

are plain numbers and we named them "cont1, cont2, stop1, stop2" for better intuition as in Ex. 3.3.23. According to Fact. 3.4.3 we can equivalently check the causality relations for checking independence of two neighbouring transformation steps. We have that "cont1" $<_{wc}$ *"stop1" and "cont2"* $<_{wc}$ *"stop2". This means that these pairs are not independent and therefore, "cont1" has to occur before "stop1" and "cont2" before "stop2". According to Thm. 3.4.12 and Def. 3.4.10 we can check, whether the sequence* $seq(d')$ *can be derived by the presented relations. First of all, the sequence is derived by switchings of independent rule occurrences without considering NACs: stop1* $\leftrightarrow$ *cont2, stop1* $\leftrightarrow$ *stop2, cont1* $\leftrightarrow$ *cont2 and cont1* $\leftrightarrow$ *stop2 leading to* $s = \langle$*cont2;stop2;cont1;stop1*$\rangle = seq(d')$*. Now, for each rule occurrence and NAC in* s *there is a weak enabling rule before or a weak disabling rule behind or the rule disables itself. For instance for* $N_1[2]$ *of "cont1" we have "stop2" with stop2*$<_{wen[2]}$*cont1 and therefore,* $N_1[2]$ *is not present in the intermediate object. All together* $s = seq(d')$ *is a legal sequence with respect to* d*, which implies that* $seq(d') \overset{\pi}{\approx}_{\mathcal{S}} seq(d)$ *and hence,* $d' \overset{\pi}{\approx} d$ *according to Thm. 3.4.12.*

Note that a pairwise switching of the example transformation sequence with NACs is not possible, because each pair is sequentially dependent - either by causal relation or by NAC dependency. Therefore, this sequence cannot be derived by standard switching of completely independent transformation steps according to switch equivalence with NACs in Def. 3.2.5. This shows that switch equivalence with NACs based on sequential independence of transformation sequences with NACs [HHT96, LEO06, LEOP08] only leads to a subclass of equivalent transformation sequences and in general, many equivalent transformation sequences cannot be derived. But as the example transformation sequence shows, all permutation-equivalent transformation sequences are of interest, because a certain person may not be available for a concrete time slot while another person could use the time and give some support for the task.

3.5 Analysis of Permutation Equivalence Based on Petri Nets

Based on the construction of the process model of a transformation sequence given by an STS as presented in Sec. 3.2 and the derived relations according to Sec. 3.3, we now present the construction of the "dependency net" of the transformation sequence, which is given by a P/T Petri net that specifies only the dependencies between the transformation steps leaving out all further details of the involved objects of the transformation sequence. In the case of graphs we have that all details about the internal structure of the graphs and the transformation rules are excluded, allowing us to improve the efficiency of the analysis of permutation equivalence. This means that we provide an analysis of the interleaving

semantics of an $\mathcal{M}$-adhesive transformation system based on the analysis of the firing behaviour of the generated Petri net.

Given a transformation sequence d and its STS $STS(d)$, then the dependency net $DNet(d)$ of d is a P/T Petri net which contains a transition q_k for each transformation step d_k in in d, where q_k is the rule name in $STS(d)$ that corresponds to the step d_k. Thus, a firing of the net will induce a sequence of rule names for the STS. By Thm. 3.5.3 we can further ensure that this sequence specifies a transformation sequence that is permutation-equivalent to d.

The dependency net $DNet(d)$ further contains for each transition one place marked with one token and the place is in the pre-domain of the transition in order to ensure that the transition may fire at most one time. Moreover, for each element of the causality dependency relations there is a place that ensures the induced partial order given by the causal dependencies. Finally, there are places that specify the inhibiting effects of the NACs. The marking of a NAC place defines the absence of parts of the forbidden pattern. Thus, if a NAC place is empty, the transformation sequence that corresponds to the current firing sequence of the net will show an object that completely includes the NAC pattern and the rule, which owns the NAC will not be applicable. Correspondingly the empty NAC place deactivates the transition that simulates this rule. This means that the inhibiting effects of the NACs do not require to use inhibitor nets as process model, such that efficient analysis techniques available for standard P/T nets are applicable.

Definition 3.5.1 (Dependency Net $DNet$ of a Transformation Sequence)**.** *Let d be a NAC-consistent transformation sequence of an $\mathcal{M}$-adhesive TS, let* $\mathrm{STS}(d) = (T, P, \pi)$ *be the generated STS of d and let $s = seq(d) = \langle q_1, \ldots, q_n \rangle = \langle 1, \ldots, n \rangle$ be the sequence of rule names in* $\mathrm{STS}(d)$ *according to the steps in d. The dependency net of d is given by the marked Petri net $DNet(d) = (N, M)$, $N = (PL, TR, pre, post)$, defined as follows:*

- $TR = \{q_k \mid k \in \{1, \ldots, n\}\}$
- $PL = \{p(k) \mid q_k \in TR\} \cup \{p(j <_x k) \mid q_j <_x q_k \wedge x \in \{rc, wc, d\} \cup \{p(k, N[i]) \mid N_k[i]$ *is a NAC of* q_k *in* $STS(d) \wedge q_k \not<_{wdn[i]} q_k\}$

- $pre(q_k) = p(k) \oplus \sum_{\substack{q_j <_x q_k \\ x \in \{rc,wc,d\}}} p(j <_x k) \oplus \sum_{\substack{q_j <_{wdn[i]} q_k \\ j \neq k}} p(j, N[i]) \oplus \sum_{p(k,N[i]) \in PL} p(k, N[i])$

- $post(q_k) = \sum_{\substack{q_k <_x q_l \\ x \in \{rc,wc,d\}}} p(k <_x l) \oplus \sum_{q_k <_{wen[i]} q_l} p(l, N[i]) \oplus \sum_{p(k,N[i]) \in PL} p(k, N[i])$

- $M = \sum_{q_k \in TR} p(k) \oplus \sum_{\substack{q_j <_{wdn[i]} q_k \\ p(j,N[i]) \in PL}} p(j, N[i])$

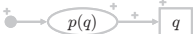
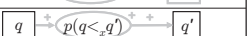
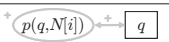
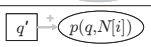
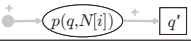
STS(d) = (T,P,π)	$DNet(d) = ((PL,TR,pre,post),M)$
1. For each $q \in P$	
2. For all $q,q' \in P$, $q <_x q'$, $x \in \{rc,wc,d\}$	
3. For all $q \in P$ with $q \not<_{wdn[i]} q$, $i \in \mathbb{N}$	
a) N[i] of q	
b) For all $q' \in P$: $q' <_{wen[i]} q$	
c) For all $q' \in P$: $q <_{wdn[i]} q'$	

Figure 3.13: Visualization of the construction of the dependency net

Fig. 3.13 shows an algorithm how the steps of the construction of the dependency net are performed. The construction steps are performed in the order they appear in the table. Each step is visualized as a rule, where gray line colour and plus-signs mark the inserted elements. The matched context that is not changed by a rule has black line colour, e.g. in step two the new place "$p(q <_x q')$" is inserted between the already existing transitions q and q'. The tokens of the initial marking of the net are represented by bullets that are connected to their places by arcs. In the first step each rule of the STS is encoded as a transition and it is connected to a marked place for ensuring that it cannot fire twice. In step 2, between each pair of transitions in each of the relations $<_{rc}$, $<_{wc}$ and $<_d$, a new place is created in order to enforce the corresponding dependency. The rest of the construction is concerned with places which correspond to NACs and can contain several tokens in general. Each token in such a place represents *the absence* of a piece of the NAC; therefore if the place is empty, the NAC is complete.

In this case, by step (3a) the transition cannot fire. Consistently with this intuition, if $q <_{wen[i]} p$, i.e. transition q consumes part of the NAC $N[i]$ of p, then by step (3b) q produces a token in the place corresponding to $N[i]$. Symmetrically, if $q <_{wdn[i]} p$, i.e. p produces part of NAC $N[i]$ of q, then by step (3c) p consumes a token from the place corresponding to $N[i]$. Notice that each item of a NAC is either already in the start graph of the transformation sequence or produced by a single rule. If a rule generates part of one of its NACs, say $N[i]$ ($q_k <_{wdn[i]} q_k$), then by the acyclicity of $STS(d)$ the NAC $N[i]$ cannot be completed before the firing of q_k: therefore we ignore it in the third step of the construction of the dependency net.

Note that the constructed net in general is not a safe one, because the places for the NACs can contain several tokens. Nevertheless it is a bounded P/T net. The bound is the maximum of one and the maximal number of adjacent edges at a NAC place minus two.

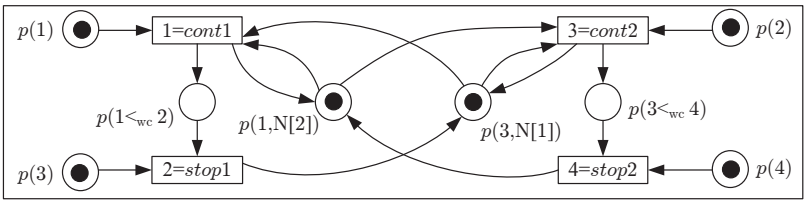


Figure 3.14: Dependency Net $DNet(d)$ as Petri Net

Example 3.5.2 (Dependency Net)**.** *Consider the transformation sequence d from Ex. 3.2.1 and its derived STS in Ex. 3.3.23. The marked Petri net shown in Fig. 3.14 is the dependency net $DNet(d)$ according to Def. 3.5.1. The places encoding the write causality relation are "$p(1 <_{wc} 2)$" and "$p(3 <_{wc} 4)$". For the NAC-dependencies we have the places "$p(1, N[2])$" for the second instantiated NAC in the first transformation step of d and "$p(3, N[1])$" for the third transformation step and its first instantiated NAC. The other two instantiated NACs are not considered, because the corresponding rules are weakly self-disabling ($q <_{wdn[i]} q$). At the beginning the transitions $cont1$ and $cont2$ are enabled. The firing sequences according to the transformation sequences d and d' in Fig. 3.7 can be executed and they are the only firing sequences of this net. Thus, the net specifies exactly the transformation sequences which are permutation-equivalent to d.*

We now show by Thm. 3.5.3 below that we can exploit the constructed Petri net $DNet(d)$ for a transformation sequence d to characterise all transformation sequences that are permutation-equivalent to d, by analysing the firing behaviour of $DNet(d)$. Note that according to Def. 3.5.1 each sequence s of rule names in the STS $STS(d)$ can be interpreted as a sequence of transitions in the derived marked Petri net $DNet(d)$, and vice versa. This correspondence allows us to transfer the results of the analysis back to the STS. More precisely, we can generate the set of all permutation-equivalent sequences by constructing the reachability graph of $DNet(d)$, which therefore can be considered as a compact representation of this equivalence class.

Recall that a *transition complete firing sequence* of a Petri net is a firing sequence where each transition of the net occurs at least once; notice also that in a dependency net according to Def. 3.5.1, each transition can fire at most once by construction. This means in our case each transition fires exactly once. The following Thm. 3.5.3 presents a sound and complete analysis of permutation equivalence by complete firing sequences in the corresponding

dependency net. This way, we provide a behaviour analysis technique concerning the interleaving semantics of an $\mathcal{M}$-adhesive transformation system based on low-level Petri nets.

Theorem 3.5.3 (Analysis of Permutation Equivalence Based on Petri Nets)**.** *Let d be a transformation sequence with NACs in an STS-compatible $\mathcal{M}$-adhesive transformation system, let $Prc(d) = (\mathcal{S}, v)$ be the process of d and let $DNet(d)$ be its dependency net. Then a transformation sequence d' is permutation-equivalent to d ($d' \overset{\pi}{\approx} d$) if and only if the sequence of rule names $s_{d'}$, which contains all the transformation steps of d in the order they are actually executed in d', is a transition complete firing sequence of the marked P/T Petri net $DNet(d)$.*

Proof (Sketch). Let d be a transformation sequence with NACs, $STS(d)$ be its derived STS and $DNet(d)$ be the constructed dependency net. We can interpret a transition complete firing sequence s of $DNet(d)$ within the STS $STS(d)$ and show that it corresponds to a valid derivation in $STS(d)$. This allows us to use Thm. 1 in [Her09a] showing that the transformation sequence derived from s is permutation-equivalent to d. Vice versa, given a transformation sequence d', which is permutation-equivalent to d, we can show that the corresponding sequence $s_{d'}$ is a transition complete firing sequence in $DNet(d)$. For a complete proof see the proofs of Thms. 1 and 2 as well as the derived Cor. 1 in [HCEK10b]. □

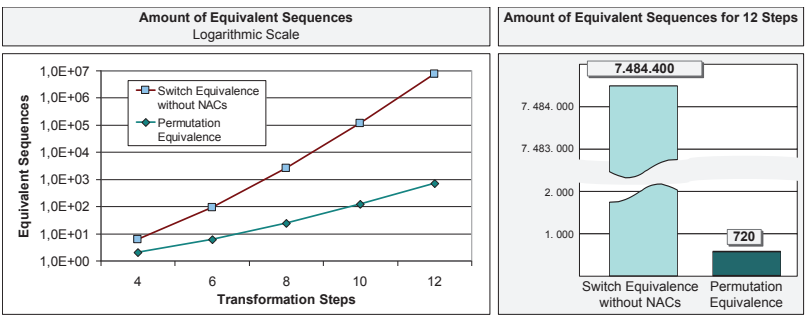


Figure 3.15: Comparison of the Amount of Equivalent Sequences

Besides soundness and completeness of the analysis as presented before we now focus on its efficiency. Therefore, we reconsider the extended example in Ex. 3.2.7 of Sec. 3.2 and compare the analysis efforts of the new technique with those of a direct analysis of the transformation sequence. This comparison shows a significant advantage of the technique and the effect is not limited to specific examples. The benefit is high for transformation

sequences, where many steps overlap on matches and include dependencies because of NACs.

Let us consider to perform a direct analysis based on the definition of permutation equivalence. We call this the brute force variant, where first all switch-equivalent transformation sequences are generated without considering the NACs, and then those which do not respect the NACs are filtered out. This means that we only have to respect the causality between the first and the second step of each of the 6 blocks. Therefore, we can always switch neighbouring steps of different blocks. As explained in Ex. 3.2.7 there are $6! = 720$ permutation equivalent transformation sequences and $6! \times F = 12!/2^6 = 7.484.400$ switch equivalent transformation sequences with $F = 10.395$. Since the complexity of a function, which is dominated by a factorial expression, is super-exponential, the calculation of invalid sequences should be avoided in general.

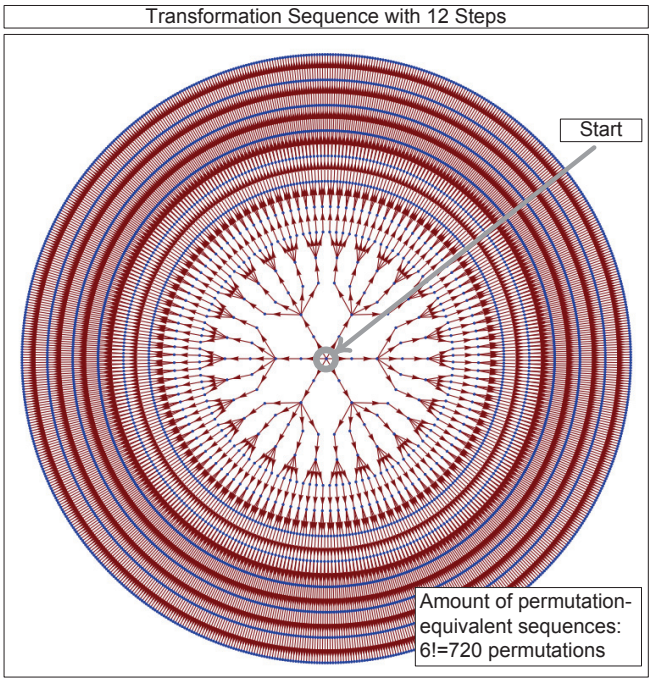


Figure 3.16: Reachability Graph of the Dependency Net for 12 Steps

Figure 3.16 shows the reachability graph of the dependency net, which consists of 12 transitions and 48 places. The graph was generated using the tool AGT-M, which is presented in Ch. 7. As shown before, the reachability graph contains $6! = 720$ paths specifying the different permutations and each path ends at a separate leaf node, such that each leaf node marks one specific permutation-equivalent transformation sequence. The reachability graph shows the symmetric nature of the specific example and future work will encompass symmetry reduction techniques, such that the size can be reduced for examples with symmetry.

Now, let us compare the effort of generating the set of permutation-equivalent transformation sequences using the dependency net $DNet(\widetilde{d})$ with the effort of a brute force generation directly based on Def. 3.2.6. The result according to Ex. 3.5.4 below is that a lower bound for the effort of the brute force variant is still about 3 orders of magnitude higher than an upper bound for the analysis based on the dependency net.

		Amount of operations (one per arc and place)		Amount of remaining rules	
Step	Nodes in RG	Update of Markings	Checks of activated Transitions	contx	stopx
0	1	-	288	6	6
1	6	42	240	5	5
2	6	42	1.200	5	5
3	30	210	960	4	4
4	30	210	3.840	4	4
5	120	840	2.880	3	3
6	120	840	8.640	3	3
7	360	2.520	5.760	2	2
8	360	2.520	11.520	2	2
9	720	5.040	5.760	1	1
10	720	5.040	5.760	1	1
11	720	5.040	-	0	0
12	720	5.040	-	0	0
	Sum:	27.384	46.848		
	Sum of binary operations:		74.232		

Figure 3.17: Efforts for Constructing the Reachability Graph

Example 3.5.4 (Analysis Efforts)**.** *This reachability graph has 3.913 nodes and 3.912 edges, and an upper bound for its computation can be derived as follows. During the generation of $RG(DNet(\widetilde{d}))$, we store the transitions that have fired, since the maximum of one time firing is ensured by definition. At each node, we check the activation condition of each transition that has not fired before, i.e. for each "$contx$" transition we have to check 1 + 5 =6 pre arcs and for each "$stopx$" transition we have to check $1 + 1$ pre arcs. After checking the activation, we continue by updating the marking, which means to*

perform 1+5+1=7 token modifications for "contx" and 1+1+5=7 token modifications for the case that a "stopx" transition was fired.
The calculated numbers at each level of the reachability graph are listed in the table of Fig. 3.17, and we derive an upper bound for the effort eff of constructing $RG(DNet(\widetilde{d}))$ given by: $eff \leq 74.232 < 9 \cdot n$ basic operations (assignments and comparisons), where $n = 12 \cdot 6! = 8.640$ is the number of transformation steps in the set of all permutation-equivalent transformation sequences. Considering the brute force variant we would have to construct $F = 10.395$ times as many transformation sequences as the number of permutation-equivalent transformation sequences. If we count the effort for constructing each transformation step including the check of NACs by only one basic operation we derive the following very lower bound for the brute force effort EFF: $F \cdot n \leq EFF$. In comparison we have:

$$\boxed{eff < 0,0009 \times EFF.}$$

Of course, the effort for constructing the Petri net has also to be taken into account, but it does not change the result. In general, the construction of the process $STS(d)$ with its relations is shown to be of polynomial time complexity with respect to the length of the transformation sequence d [Her09a]. Furthermore, the construction of the dependency net is linear with respect to $STS(d)$ with its dependency relations and in this this example there are only 48 places in the constructed Petri net. Note that still all steps in $\widetilde{d}$ are sequentially dependent with the NACs and therefore, no direct switching is possible.

Summing up, the new notion of permutation-equivalence is exactly the notion of equivalence needed for transformation systems with NACs and coincides with switch equivalence for systems without NACs. Since permutation equivalence is by definition a restriction of switch equivalence without NACs a direct analysis of a given transformation sequence by first generating all possible switchings without respecting NACs and then filtering out the inconsistent ones is to inefficient as illustrated by the benchmark. The compact process model of a transformation sequence given by its derived STS builds the basis for an efficient analysis, where the costs for pattern matching are reduced massively and avoided completely during the analysis once the STS is constructed. Matches are given by explicit inclusions into the superobject.

By Thms. 3.4.12 and 3.5.3 we have shown that the analysis of permutation equivalence based on the derived STS as well as based on the more compact dependency net are sound and complete. While the dependency net specifies the possible sequences of steps only, the corresponding STS provides the information for deriving the actual permutation equivalent transformation sequences to the sequences generated by the dependency net. The technique furthermore, allows for partial analysis in the case that only some permutation-equivalent transformation sequences are required or a specific reordering shall be checked to specify a permutation-equivalent transformation sequence.

Chapter 4

Behaviour Analysis and Optimization of Visual Enterprise Process Models

Business process models are developed and maintained in order to analyse and improve running processes of an enterprise as well as to specify new processes that shall be implemented and executed later on, where quality and efficiency measurements may be performed on the models already. Business process modelling itself can be seen as a process, because business processes need to be updated during the whole lifecycle of an enterprise. For this reason, the development of business process models is placed within a reengineering cycle [Off97] starting with the identification of the concrete process which is thereafter analysed, such that the possibly existing process models can be updated. Based on these results the process is modified, tested and implemented again. Thus, the next reengineering cycle can start with the new concrete process whenever there is a need from the business point of view. Clearly, the additional modelling efforts need to pay off. Besides an increased efficiency of the processes this can be achieved by an increased quality and risk control, in particular with respect to business continuity management.

This chapter shows how the formal techniques of Ch. 3 can be used to analyse business process models concerning equivalent modifications and an efficient management of business continuity based on intuitive continuity snippets that specify alternative fragments of the process model for cases of failures. For this purpose we present in Sec. 4.2 a formal operational semantics for EPC models and show how functional and security requirements can be validated. As main results we show by Thms. 4.2.7 and 4.2.10 how equivalent and valid process runs can be generated from the standard process together with its continuity snippets. Furthermore, we illustrate the application of the analysis and generation techniques for the case study based on prototypical tool support in Sec. 4.3. Moreover, the formal foundation of EPC business process models in this chapter enables the formal analysis of business process models concerning their conformance to business service structure models in Ch. 6.

4.1 Business Process Modelling

Quite a number of different modelling languages exist for business processes, like e.g. the business process modelling notation [OMG09] by Object Management Group (OMG). In order to execute BPMN models several approaches have been presented, which transform BPMN models into executable models using, e.g., the Web Services Business Process Execution Language (WS-BPEL) [OAS07, ODtHvdA06, BE09].

The business process modelling language of Event Driven Process Chains (EPCs) as part of the ARIS framework [SN00, IDS10] has been used already for a relatively long time and at many enterprises. The ARIS tool suite supports both, EPC models and BPMN models. Depending on the domain specific modelling purposes and the given requirements for the respective business processes both languages can be used simultaneously or exclusively, respectively.

In the case study of this chapter we focus on an EPC model that shows a simplified loan granting process at a financial institution, which was created and analysed within a fruitful research project in cooperation with Credit Suisse. Since EPCs provide a very intuitive notation and they are widely used, it was quite natural for us to chose them as basis for the development and application of formal analysis techniques. However, in order to provide important new analysis techniques concerning equivalent process modification and business continuity management, we slightly extend the language of EPCs.

The next section introduces the case study of a loan granting process based on a concrete EPC model. Thereafter, Sec. 4.1.2 describes the main challenges within business continuity management and the derived requirements for business process. Finally, Sec. 4.1.3 discusses main problems in business process modelling.

4.1.1 Event Driven Process Chains

In the case study of this chapter we analyse a generalized and partly simplified loan granting process as it is executed at financial institutions like, e.g., Credite Suisse. The business process is modeled as an extended event driven process chain (EPC). Based on the process model for the standard case we present in Sec. 4.2 and Sec. 4.3 analysis and generation techniques to support optimization and business continuity management.

The process model shown in Fig. 4.2 was presented in [BHE09a] and provides detailed information for the execution and automated support by a workflow engine using a slightly modified and extended version of event driven process chains (EPCs) [Sch01] called workflow engine based and data-flow oriented EPC (WDEPC) which we introduced in [BHE09a]. EPCs are chosen as domain language for process models, because they are widely used in the financial domain and in particular at Credite Suisse, which supported the

joint research. Thus, the close relationship to EPCs will serve to keep the models intuitive for the modellers that are designing and checking the business process models.

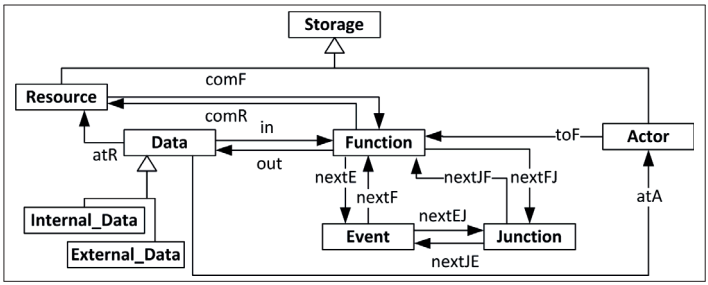


Figure 4.1: Meta model of WDEPC diagrams (extended EPCs)

The meta model for the abstract syntax of WDEPCs is shown in Fig. 4.1 and can be equivalently seen as a type graph TG_{Meta}, such that the abstract syntax graphs of WDEPC models are typed over TG_{Meta}. Each WDEPC model may contain business functions, events, organizational entities, data objects, and applications executing business functions like in usual EPCs. Furthermore, there are edges for the specification of data flow between business functions and data storage units. Such storage units are either resources, like databases and paper prints, or, alternatively, actors (organizational entities), like concrete persons or executable applications. The data flow between these elements is specified using the edge types “in”, “out”, “atR” and “atA”. This way, the execution of a process on a workflow engine can be mainly controlled by the data flow, where electronically processed data is temporarily cached in a local storage of the engine.

Moreover, WDEPC models have to satisfy the following well-formedness constraints, in order to avoid ambiguity. There are no parallel edges of the same type. Each data node is either an input data or an output data node, i.e. it has either one adjacent edge of type “in” or one of type “out”.

One important security requirement for the considered loan granting process is the separation of the following parts: the customer relationship management process, the credit approval and administration process and the credit monitoring process. For this reason, the acting employees in this process have to be distinguished. The client relationship manager (RM) is an expert in the concrete client relationship and executes the main part of the customer service. The credit advisor (CA) is an expert in the portfolio of credit products and services which are offered by the financial institution and the credit officer (CO) is charged with the responsibility of approving credit transactions. The process itself covers the whole lifespan of a granted loan, starting with the demand for a loan, ending with its finished payment plan.

The main components of the process are given by the following list:

1. Initiating the customer relation and inquiry of relevant data
2. Evaluation of customer worthiness and decision of customer acceptance (process termination if customer is not accepted)
3. Product customization, contract approval and contract conclusion
4. Payment, loan repayment and closure

The WDEPC process model of the loan granting process of the case study is shown in Fig. 4.2 in concrete syntax. From left to right the diagram is arranged in columns consisting of data storage objects (e.g. documents, data base servers and actors), the data objects (depicted as light blue rectangles), business functions (green rounded rectangles), events (red hexagons) and, finally, on the right most column there are actors (yellow ellipses) as well as acting applications (white rectangles) that execute the business functions. The execution of an EPC is performed along the sequence of events and some additional control elements like forks and joins (grey circles).

Once a customer (C) arrives at the bank, he will be interviewed by the relationship manager (RM) (functions F1-F4 in the workflow model in 4.2). At first, he has to give personal information and the relationship manager will try to make a first estimation about the possible customer value in an assumed business relationship. Thereafter, the demand of the client is recorded and all data is entered into IT systems by the relationship manager. Afterwards, the credit worthiness of the client is approximated and a customer rating is calculated automatically by two different applications (F5-F6). Based on the computed rating the credit advisor decides whether the client is accepted for a loan or not (F7). In the case of acceptance, an optimized product is created by the credit advisor and the relationship manager (F8). Based on this optimized products the relationship manager creates a contract (F9). This contract is then signed by the customer, the relationship manager and the credit officer (CO) who approves the contract (F10-F12 in Fig. 4.2). This way, the four-eye principle – as one typical security requirement – is respected, because the because the relationship manager and the credit officer must not be identical for one concrete execution of the process. As soon as the contract is approved the bank pays the granted loan to the client and the client starts the repayment according to the payment plan of the contract. The contract is closed after the payment plan is completely fulfilled by the client (F13-F15).

Within the WDEPC process model, the data-flow is specified explicitly by edges between the data objects and the resources. These information are used in Sec. 4.2.2 for the generation of equivalent process variants using the induced data dependencies. In order to specify which data objects of the process are temporarily stored by the workflow engine the rectangles have either a solid thick line or, for the case that they are stored externally,

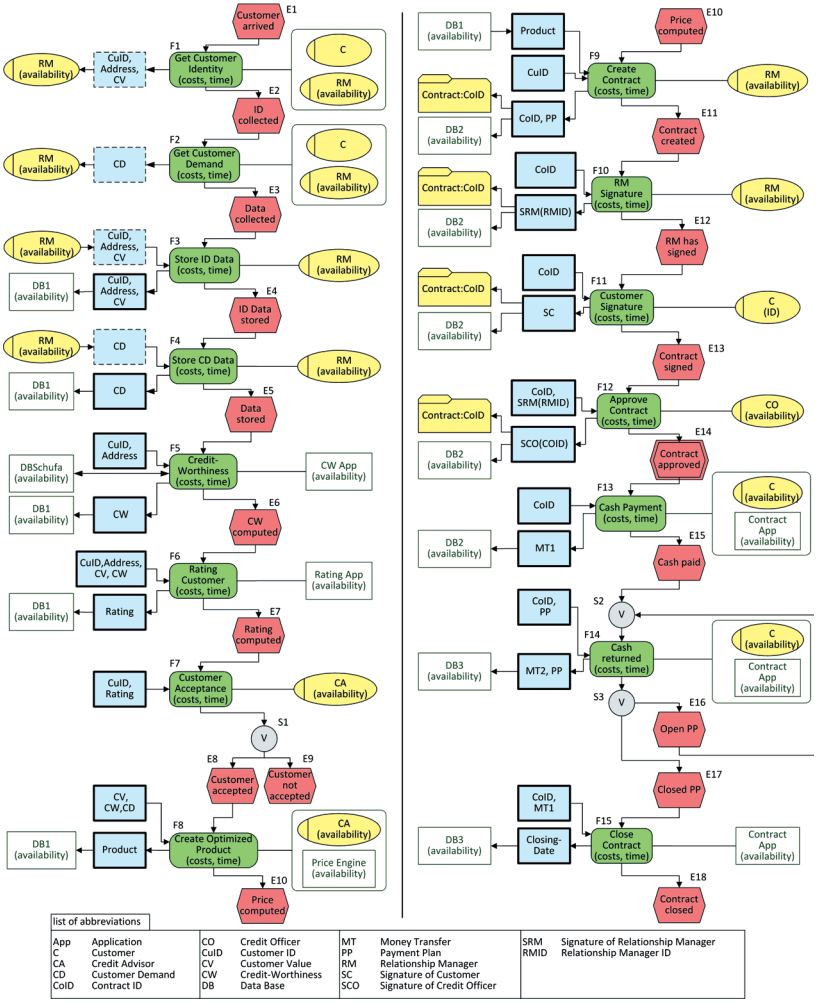


Figure 4.2: WDEPC workflow model for the loan granting process

they have a dashed line. This allows us to cover automated and non-automated parts of a business process by one single human-centric business process model as it was requested by Credit Suisse. The data which is cached by a workflow engine can be reused if needed.

4.1.2 Business Continuity Management

Business continuity management concerns all kinds of failures that may occur during the execution of business processes. Since such failures can directly affect business profits and moreover, business continuity in general, the importance of an efficient as well as ensured business continuity management is high, in particular for organizations and institutions in the financial sector, but also for those active in the telecommunication, transport or public area. This section discusses main problems within business continuity management with a special focus on financial institutions based on the fruitful cooperation with University of Luxembourg and Credit Suisse [BHE09a, BHG11], in which we discussed problems for concrete and realistic scenarios, e.g. the simplified loan granting process presented in Sec. 4.1.1. Based on the formal results in Ch. 3 we provide powerful analysis and generation techniques for these problems in Sec. 4.2-4.3.

From the general point of view, a business continuity management is based on a business continuity plan, which is applied after a critical business process is disrupted. The business continuity plan has to ensure at least a minimum level of business activity, such that the survival of the enterprise is guaranteed during the execution of the business recovery process. In addition to the functional requirements on business processes that are essential and necessary, there are usually further non-functional requirements that have to be respected by the standard process and by the continuity processes. These requirements are given by e.g. legislative or enterprise specific regulations. Especially in the case of business continuity management at financial institutions there are several security requirements, which are defined independently by different parties, such as legislation or standardization institutions.

The main aim of business continuity management is to control all critical business processes of an organization in a way that ensures business continuity in the case that some these processes fail. In the worst case, the combination of some failures of critical business processes can be the cause that the company goes out of business. Even if certain failures occur, the business has to be continued, i.e. alternative business processes must run and ensure correct but possibly less efficient business execution. Stopping the business is no option, because of high penalty costs, high losses, high fix costs, and a dramatic decrease of customer acceptance. This means that an effective business continuity management is not sufficient if only some continuity plans are developed independent from each other concerning single failures only. An effective business continuity management has to support the management of combination of several failures occurring at the same time.

In order to evaluate and certify the quality of existing business continuity management new standards were developed, like the British Standard BS 25999 (BS 25999-1:2006 [Bri06] and BS 25999-2:2007 [Bri07]) on which the the German BSI Standard 100-4 [Fed09] is based. However, according to [Boe09], the survival of an enterprise in the event

of a disaster is not ensured mainly by the pure existence of a business continuity management system according to BS-25999, but by the output and efficiency of the specified concrete business continuity plans and disaster recovery plans. Furthermore, the different regulations the enterprise has to comply to, like e.g. Basel II [Bas06] or the Sarbanes-Oxley Act [Uni02], induce further constraints on the business processes in general including the business continuity processes.

From the point of view of Credit Suisse the simplified loan granting process model LG presented in Sec. 4.1.1 is critical and several failures can occur. Here, we focus on the availability of organizational agents, IT applications and resources. We will show how additional process fragments, called continuity snippets, can be used to generate and provide back-up solutions, where parts of the standard process are replaced by the snippets. The snippets can be kept small and simple, which reduces the efforts for continuity modelling. They are automatically combined in Sec. 4.3.1 using the formal analysis techniques of Ch. 3.

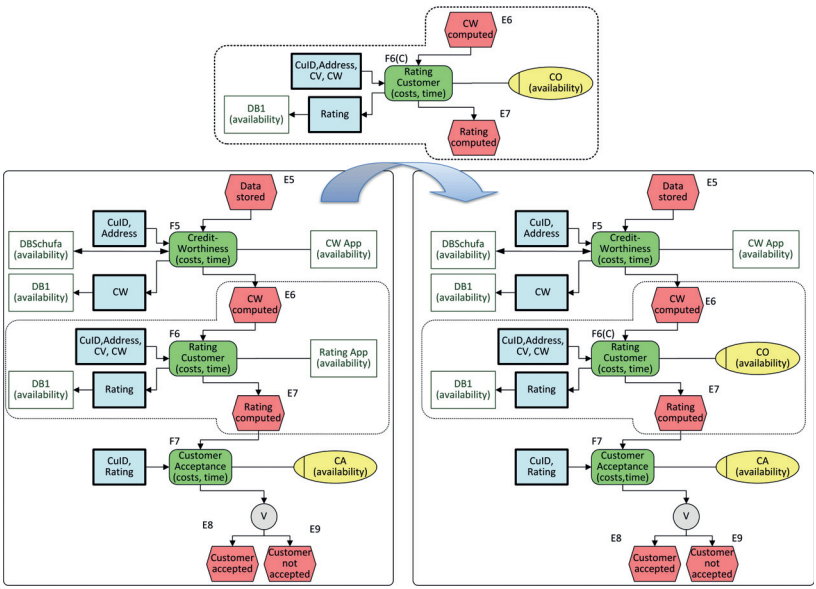


Figure 4.3: Alternative Continuity Snippet "Rating Customer" (1 of 16 simple snippets) and Part of the Alternative Workflow

The lower part of Fig. 4.3 shows a continuity snippet (F6(C)) for the function *rating customer* (F6). In the case that the *rating application* that executes the business function

becomes unavailable the process fragment with function (F6) can be replaced by the continuity snippet with function (F6(C)) where the *customer officer* will execute the business function *rating customer*. Note that in this special case the start event and the end event of both fragments are equal. However, this strong requirement is not needed for the general case and we will show in Sec. 4.3.1 how fragments can be inserted in a relaxed way such that the control events and data dependencies between the fragments are respected.

Definition 4.1.1 (Continuity Snippet)**.** *A WDEPC* snippet *is a small WDEPC model consisting of one business function node and its context elements, which are the connected actor nodes, event nodes and data nodes with its storage nodes (actor or ressource nodes). A* continuity snippet *for a WDEPC model M contains a function $(F\ x(C\ y))$ for a function (Fx) in M, where $x, y \in \mathbb{N}$.*

The idea of modelling continuity snippets instead of complete continuity processes was motivated by organizational requirements at Credit Suisse where knowledge about continuity scenarios is not necessarily provided by the same people that manage the standard business processes and where knowledge is distributed among different persons. The advantage of this approach is the flexibility concerning local changes leading to less efforts for maintenance during model and enterprise evolution. The complete set of continuity snippets – in sum 16 – is presented in [BHG10].

While the given process model shows a fixed order of certain functions, many of them are not directly dependent on each other. But they are partly dependent on the data that is produced by previous steps. Look e.g. at functions “F1” to “F4” of the WDEPC LG as depicted again in Fig. 4.4, where customer data is retrieved and stored. Clearly, function “F1” needs to be executed before “F3” and similarly “F2” before “F4”, but there are no further dependencies. Thus, leaving out the synthetic events in between we are interested in all permutations of such steps fulfilling the data dependencies to generate valid process variants. However, there are some specific control events like “E8”, “E9” or “E14” which restrict possible permutations in the sense that they play the role of a boundary which cannot be passed when moving up or down. Critical parts of the workflow are e.g. at functions “F10” where the contract is signed by the relationship manager and “F12” where it is approved by the credit officer. Here, the four-eye principle applies as a security requirement and demands that these functions have to be performed by two different persons. For this reason, the continuity processes have to ensure this requirement, too. This requirement is formalized as graph constraint and checked during the generation of process variants in Sec. 4.3.

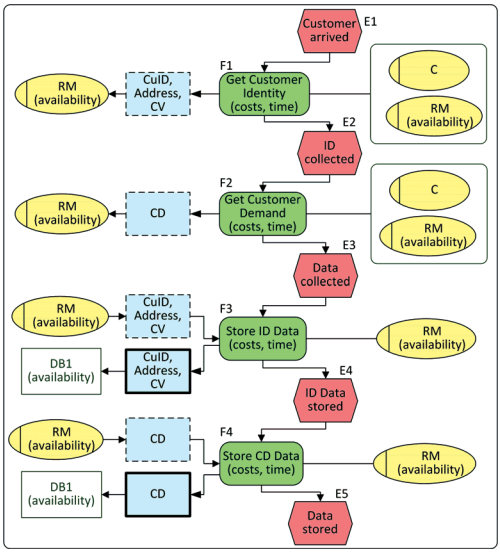


Figure 4.4: First four steps of the WDEPC model LG

4.1.3 Problems of Business Process Modelling

The main aim of business process modelling is to design, control, manage and evaluate already running and future business processes. Semi automated analysis techniques for the evaluation, simulation and verification of process models as well as for the generation of derived process models for business continuity management can show high impact for the effectiveness and efficiency of the whole modelling process. For this reason, this section first discusses the main problems of business process modelling with a special focus on business continuity management. We then present formal and efficient techniques for solving these problems in a suitable way and evaluate in Sec. 4.3.2 the quality and effectiveness of the developed techniques.

According to Tab. 4.1 below, we detected the following main problems and propose some modifications for improving the quality and maintenance of the process models. While enterprises in the industrial production segment maintain and run their modelled production processes and in particular their supply chain management by automated enterprise software suites, this is not the case for the more organizational business process, which are of main importance for financial institutions. In this case, process models are partly incomplete and do not cover all processes. The models are given partly by human-centric

Table 4.1: Problems and goals for business process modelling

Criterion	Common Practice	Goal
1. Form of Process Models	Incomplete, human-centric visual notion, additional text documents	Comprehensive, extensive, human-centric visual notation, decreased amount of text documents
2. Form of Requirements	Mainly text documents	Text documents plus formalized constraints
3. Form of Continuity Plans	often plain text documents	compact and concise models
4. Evaluation of Continuity Plans	usually manual without formal verification of requirements	semi automated, formally verified results ensuring compliance
5. Maintenance	manual, modifications of standard process require modifications of continuity plans	improved maintainability, continuity plans are partly updated automatically
6. Consistency	mainly checked manually, continuity processes contain redundant information from standard processes	partly automated formal analysis, reduced redundancies

visual models, like EPCs or BPMN models together with additional plain text documents. The goal in this context is to improve business process modelling to derive comprehensive models that cover the main processes and to reduce the amount of plain text documents.

In order to check the functional and non-functional requirements, they should be formalized in a significant amount, such that formal automated techniques can be used for validation, which reduces the manual efforts and possible errors. Especially financial institutions are faced with a high amount of non-functional requirements that need to be respected.

The modelled continuity plans are quite often not sufficiently formalized and indeed, they are usually defined by plain text documents [ZSLZ08], which causes two major problems. The first one is that maintaining and modifying these documents are complex tasks and errors are hard to detect. Therefore, continuity management can be supported by providing techniques for a semi automated generation of continuity plans based on precise and concise snippets. The second problem is that there is no adequate tool support for analysing the continuity documents with respect to their correctness, consistency and their conformance to the given functional and non-functional requirements. For this reason, the

quality of these documents cannot be ensured by formal techniques, but only by manual evaluation.

4.2 Operational Semantics and Analysis of Business Process Models

In order to analyse WDEPC business process models this section first defines a formal operational semantics using the concepts of graph transformation and subobject transformation systems as presented in Ch. 3. Furthermore, we show how functional and non-functional requirements are formalized and checked verified for concrete process executions.

Based on the formal operational semantics, which is directly generated from the abstract syntax of a WDEPC model, we present in Sec. 4.2.2 how process runs are computed and analysed with respect to their equivalent process runs. This analysis includes the combination of parts of the standard process model with the additional continuity snippets for supporting business continuity management. As main results we show by Thm. 4.2.7 that the notion of permutation equivalence introduced in Sec. 3.2 leads to the complete set of equivalent process runs to a given one. Furthermore, we show by Thm. 4.2.10 that the equivalence preserves the validity of process runs concerning the functional and non-functional requirements, where we mainly focus on security requirements as non-functional requirements.

The analysis results of this section build the basis for the generation of equivalent standard process runs and continuity process runs for different combinations of possible failures in Sec.4.3. The proposed operational semantics and analysis techniques for WDEPC models, however, are of general nature and can be applied to other process modelling languages as well with minor changes.

4.2.1 Operational Semantics Based on Graph Transformation

Business process models given by EPCs often consist of chains of functions. The intermediate events imply virtual dependencies of consecutive functions, even if these dependencies do not exist in the real process. In the following, we describe the construction of a graph grammar GG in order to provide a formal operational semantics of the loan granting process (LGP) modeled in Sec. 4.1.1.

Given a WDEPC model describing a business process then its operational semantics is derived by constructing a graph grammar, which directly induces a subobject transformation system according to Sec. 3.3. The construction of the operational semantics can be performed automatically based on the underlying abstract syntax of WDEPC-models. Each function in the WDEPC model corresponds to one rule, or, in the case that a junction

node like a fork occurs, the function node induces one rule for each possible continuation. The induced STS ensures that the semantics is not ambiguous in the way that data elements with the same name cannot be produced twice during an execution of the model, because each creation can correspond to different concrete values.

Functional requirements of the process are defined as additional graph transformation rules within the graph grammar. A functional requirement rule specifies which of the data elements have to be created during the execution of the process for a certain combination of control events. In the presented example we have the requirement that either the customer is not accepted or the contract shall be approved and a closing date has to be set.

Definition 4.2.1 (Operational Semantics of WDEPC models)**.** *Given the abstract syntax graph G_M of a WDEPC business process model M, then the corresponding graph grammar $GG = (TG, SG, P, \pi)$ defining its operational semantics is constructed as follows:*

- *The type graph TG contains the nodes and edges of the abstract syntax graph G_M, where nodes with same label and type that occur several times in the WDEPC are identified and occur only once in TG. Analogously, edges with the same source and target node and same type are identified and appear only once in TG.*
- *The start graph SG consists of the nodes for the actors and the resources only.*
- *The set of rules P contains one rule $p = (L \leftarrow K \rightarrow R)$ for each function node F in G_M. The left hand side L of a rule p contains the actors connected to the function node as well as the connnected input data elements with its resources and the edges between these elements. The graph $K = L$ and the right hand side R additionally contains the function node, the output data elements, and the edges that connect the nodes as given in G_M. A rule may be extended, if it is connected to a control event as specified in the next item.*
- *An event in the WDEPC M is a control event, if its frame has an additional line or the event is the successor of a junction node. If a control event is the successor of a function, then the RHS of the rule for the function is extended by this event. If a control event is a direct or indirect predecessor of a function then all rule components (L, K and R) are extended by this event. A function is an indirect predecessor of an event, if there is a path of successor edges (nextx, $x \in \{E, F, EJ, JE, FJ, JF\}$) from the event to the function in the abstract syntax graph G_M.*

Figure 4.5 shows the abstract syntax graph G_{LG} for the visual WDEPC business process model LG in Fig. 4.2. The type graph TG for the graph grammar $GG_{LG} = (TG_{LG}, SG_{LG}, P_{LG})$ is obtained from Fig. 4.5 by identifying all nodes with the same label and some of their adjacent edges according to Def. 4.2.1. The necessary identifications are visualized in Fig. 4.5 by grey fill colour for nodes and grey line colour for edges. A

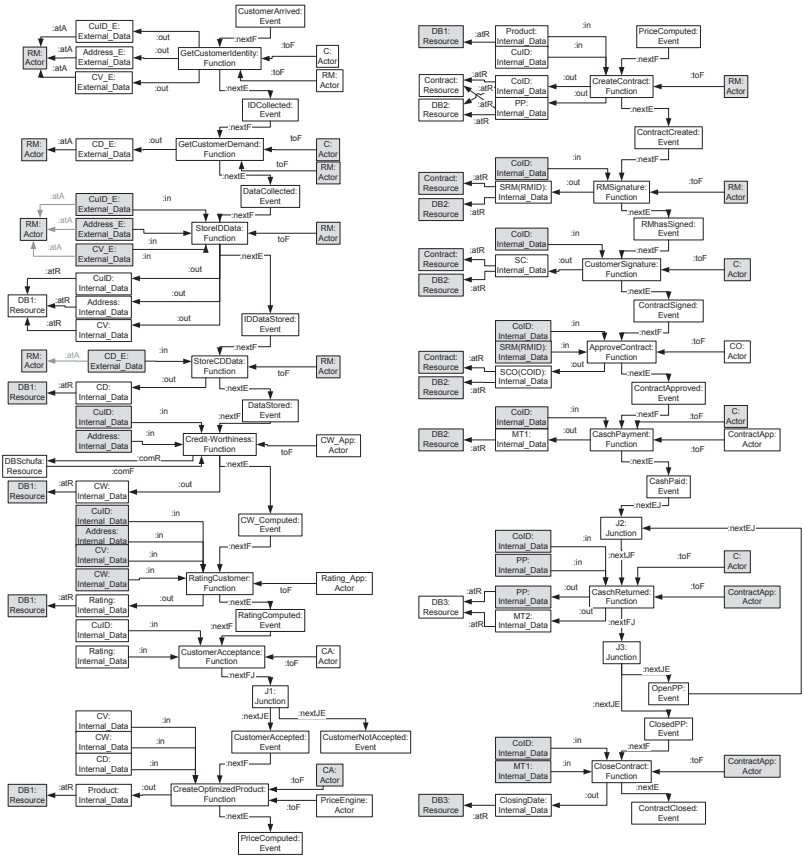


Figure 4.5: Abstract Syntax graph of the Visual WDEPC business process model LG

marked element denotes that it will be identified with an already occurring element, i.e. the marking denotes multiple occurrence. Thus, e.g. the node "RM" or the edge "atA" from "CuID" to "RM" in the top left area of the figure occur several times. In order to increase readability of the edges we only denoted their meta types and leave the explicit type name within TG given by an additional ID implicit.

Example 4.2.2 (Operational Rule)**.** *Figure 4.6 shows the rule "Store_CD_Data" of the operational semantics for the example WDEPC LG and the corresponding fragment of the WDEPC in concrete and abstract syntax. The rule is constructed according to Def. 4.2.1*

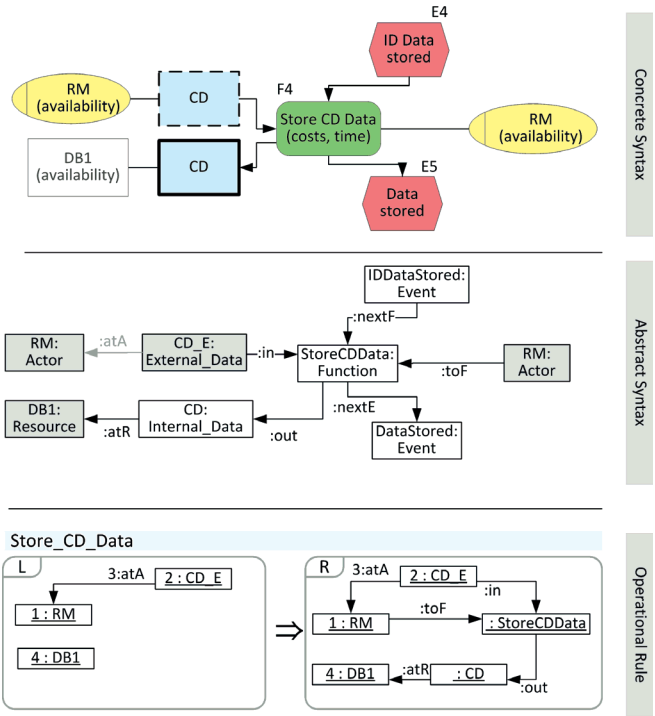


Figure 4.6: Snippet for business function "Store CD Data" in concrete/abstract syntax and corresponding rule of the operational semantics GG_{LG}

and it requires by its left hand side that the input data (CD_E, i.e. customer demand as external data) is available and was retrieved before by the relationship manager (actor "RM"). Furthermore, the database (DB1) has to be available. The effect of the rule is the creation of the function node "StoreCDData", the creation of the output data ("CD" for customer demand, which is cached and therefore internal) and the connecting edges between them.

The adjacent edges of the event nodes are not used by any rule in general, because they do not provide additional information required for the analysis. The reason is that the relevant events are contained in the same rule as the function, which is reading or creating the event.

By construction, the type graph TG of GG_{LG} corresponds to the abstract syntax of the WDEPC and is typed over the meta type graph TG_{Meta} for WDEPCs, which is shown in Fig. 4.1 in Sec. 4.1.1. From GG_{LG} we can directly derive an STS $\mathcal{S} = (T, P, \pi)$ with super object $T = TG$. Note that by construction, all rule components as well as the start graph are subobjects of the super object T.

Remark 4.2.3 (Extended Operational Semantics with Continuity Snippets)**.** *Since the additional continuity snippets (see Sec. 4.1.2) introduce some additional nodes and edges the type graph TG – therefore also the start graph SG and the super object T – are extended accordingly to contain also the additional elements. Furthermore, additional rules for the snippets are created analogous to the creation of rules for the standard WDEPC business process model.*

The graph grammar defines the operational semantics of a WDEPC in the following sense. Given a possible execution order of the business function nodes in the WDEPC then the sequence of corresponding rules for the business functions specifies an STS derivation starting at the start graph SG of the graph grammar GG_{LG}. Each intermediate graph represents the current state for the execution of the process and each rule application ensures that the necessary input data elements are available and furthermore, that they are accessible via the involved resources to which the particular actor has access.

The four rules in Fig. 4.7 are generated for the first four functions ("Get Customer Identity" $(F1)$ - "Store CD Data" $(F4)$ of the WDEPC process model. The first two functions produce data elements that are external to the workflow engine, which is specified within the rules by the suffix "-external". The rules for functions $F3$ and $F4$ automatically ensure that the involved actor can access the required data based on the edges of meta type ":atA". Thus, the generation of the rules makes explicit the dependencies between the steps and ensures for availability of the required data elements. The complete grammar containing 17 rules is given in [BHG10]. Note that the grammar forms a subobject transformation system, i.e. each STS derivation step will result in a subobject of the generated type graph TG. Therefore, the specified types of the nodes and edges within a rule uniquely determine the matching into an intermediate transformed graph.

Each rule in GG_{LG} corresponds to a function in the WDEPC LG, where some rules correspond to the same function if the function is connected to a junction element and the rules specify the different possible execution continuations at this point.

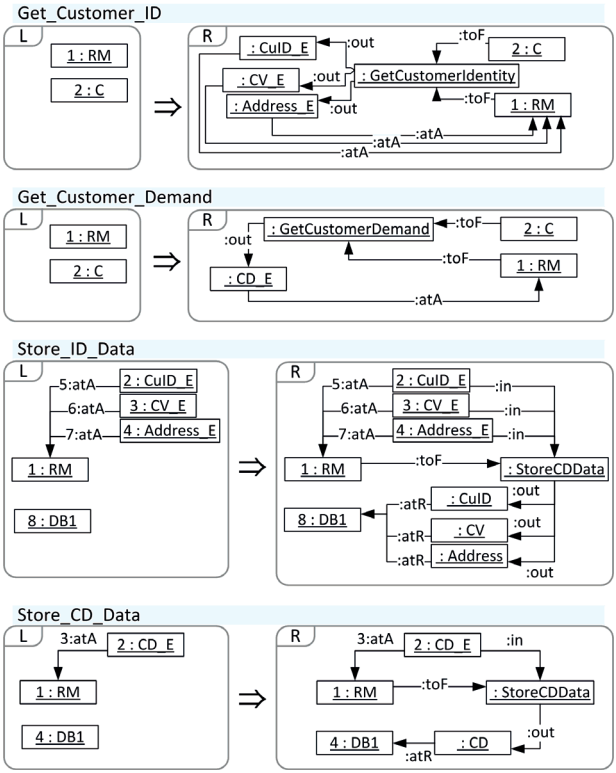


Figure 4.7: First four rules of graph grammar GG_{LG}

The loop within the WDEPC LG at function $F14$: "Cash returned" is specified within the generated grammar by two rules shown in Fig. 4.8. The first rule "CashReturnedLoop" specifies the effect of one iteration of the loop. This means that the operational semantics of the WDEPC LG given by the derived graph grammar abstracts from arbitrary amounts of loop executions and only considers one iteration, because arbitrary amounts would not allow for the generation of all process variants. An analysis concerning the termination of process components with loops can be performed separately with additional techniques like e.g. using the model transformation to the process algebra $mCRL2$ in [BHG11, BHG10]. The second rule generated from the loop is "CashReturnedEnd". The only effect of this rule is the creation of the event node "ClosedPP", which enables the execution of its successor steps.

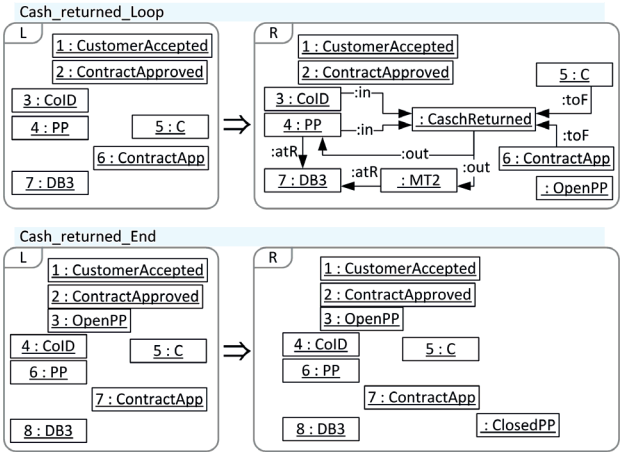


Figure 4.8: Rules of graph grammar GG_{LG} concerning the loop

A WDEPC model may be ambiguous. There may be several functions that create the same data element, i.e. which store the data element but do not read it from a resource. If there is a possible execution sequence that involves the creation of the same data element twice we consider this execution as ambiguous, because the data values may be different and the depending succeeding steps may use one or the other value nondeterministically. For this reason, our generation does only provide those execution paths in which the same data element is created only once, which is ensured by checking the dependency condition called "forward conflict".

As mentioned before, all executions can be verified to conform to the given functional and security requirements that are formalized as graph constraints. For this purpose, the corresponding STS derivation is analysed by checking that the terminal graph satisfies the graph constraints.

As an important example for a security requirement we illustrate the handling of the four-eye principle specified by the graph constraint in Fig. 4.9. Thereafter, we will focus on the functional requirements. The functions "RM Signature (F10)" and "Approve Contract (F12)" have to be performed by different persons in order to ensure a separation of concerns. The graph constraint is given by a negation of the formal constraint "samePerson", which states the following: If the premise P is fulfilled, then also the conclusion graph C has to be found, i.e. in this case the two functions are executed by the same person as specified by the connecting edges. Note that the premise graph is empty, and therefore its precondition is always satisfied.

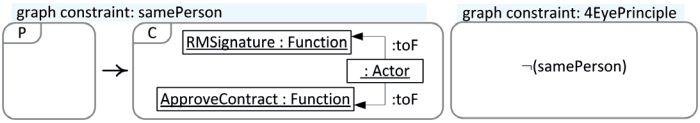
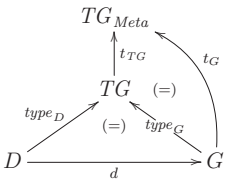


Figure 4.9: Graph Constraint: 4 Eye Principle

The shown constraint is a meta constraint, i.e. it is not fully typed over the type graph TG of the constructed graph grammar for the operational semantics. The conclusion graph C contains a node of the meta type "Actor", for which the type within TG is not specified. Thus the effective constraint is obtained by instantiating the meta constraint into a set of graph constraints typed over TG. This construction is introduced by the following definition using the notion of instantiated graphs, which corresponds to Def. 3.3.13 in Ch. 3.

Definition 4.2.4 (Instantiated Graph)**.** *Given a typed graph (G, t_G) typed over TG_{Meta}, a type graph (TG, t_{TG}) typed over TG_{Meta}, and a partial mapping from G to TG via the span of injective typed graph morphisms $G \xleftarrow{d} D \xrightarrow{type_D} TG$ typed over TG_{Meta}. An instantiation of G within TG is given by $(G, type_G : G \to TG)$, where $type_G$ is an injective graph morphism typed over TG_{Meta} and $type_G \circ d = type_D$ as shown below. The set of all instantiated graphs for G and TG via d is denoted by $Inst(d, G, TG)$.*



Using the notion of instantiated graphs we define meta graph constraints based on instantiated graph constraints.

Definition 4.2.5 (Meta Graph Constraint)**.** *Given a graph constraint $PC(a) = P \xrightarrow{a} C$ with injective a typed over TG_{Meta}, a type graph TG typed over TG_{Meta}, and a partial mapping from P to TG via a span of injective typed graph morphisms $P \xleftarrow{d_1} D \xrightarrow{d_2} TG$ typed over TG_{Meta}. The graph constraint $(PC(a), d)$ is called a meta graph constraint. An instantiated graph constraint of $PC(a)$ is given by $(P, type_P) \xrightarrow{a} (C, type_{C,i})_{(i \in I)}$, where $(P, type_P)$ is an instantiated graph in $Inst(d, P, TG)$ and $(C, type_{C,i})_{(i \in I)}$ is an ordered list of the set $Inst(a, C, TG)$ using the partial mapping from C to TG via the span of injective typed graph morphisms $C \xleftarrow{a} P \xrightarrow{type_P} TG$. The set of instantiated graph constraints is denoted by $Inst(d, PC(a), TG)$.*

A graph G typed over TG satisfies the meta graph constraint, written $G \models (PC(a), d)$, if for each instantiated constraint $(P, type_P) \xrightarrow{a} (C, type_{C,i})_{(i \in I)}$ of in $Inst(d, PC(a), TG)$ there is an $i \in I$, such that $G \models PC(a)$ with $a : (P, type_P) \xrightarrow{a} (C, type_{C,i})$.

Note that the set of instantiated graph constraints may contain several constraints but it may also be empty meaning that it is always fulfilled. For the given graph constraint in Fig. 4.9 we derive exactly one instantiated graph constraint with one conclusion as shown in Fig. 4.10.

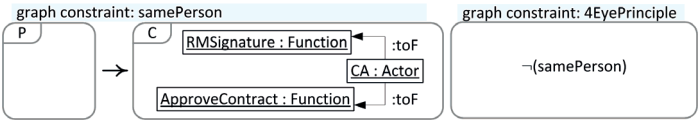


Figure 4.10: Instantiated security graph constraint for the 4-eye principle

Indeed, the complete example with all continuity snippets can lead to a situation where the four-eye principle is not respected. There are several continuity snippets for the functions “RM Signature (F10)” and “Approve Contract (F12)” and in the case that both, the relationship manager and the credit officer, are temporarily unavailable there is one combination of snippets where both actors are replaced by the credit advisor. In order to avoid such a situation we use the graph constraint and generate application conditions for the graph rules that ensure the four-eye principle.

The considered functional requirements are given by some alternative sets of required data elements and required events which specify the minimal effect of a valid execution of the process model. Each set Req induces a graph constraint $PC(a : P \to C)$ given by $P = \varnothing$ and C containing all nodes that correspond to the required elements in Req.

Since each set of required elements concerns a particular exeution of the WDEPC model, only one of the corresponding constraints has to be satisfied valid execution via the operational rules. Therefore, the combined functional requirement of the WDEPC model is a disjunction of the single functional requirement constraints.

In our case study the set of required data elements is given by $Req1 = \{CustomerAccepted, CoID, SC, ClosedPP, ClosingDate\}$ and $Req2 = \{CustomerNotAccepted\}$. In the first case the customer is accepted and the contract is created, signed as well as closed. In the second case the customer is not accepted and therefore, no additional data has to be created.

4.2.2 Equivalent Runs

Based on the formal construction of the operational semantics for business process models in the previous section we now present how these process models are analysed concerning equivalence of process executions and conformance to functional and non-functional requirements. For this purpose, we use the general analysis techniques for permutation equivalence in Ch. 3 and formalize functional requirements and security requirements by graph constraints [EEPT06]. By checking the specified graph constraints we can verify compliance of the processes with respect to the security requirements. The first main result of this section (see Thm. 4.2.7) shows how equivalent process executions are obtained and that the technique is complete in the sense that for a given execution all equivalent ones can be derived. Furthermore, for each computed process execution that satisfies the given functional and security requirements the second result given by Thm. 4.2.10 ensures that also the generated equivalent process executions satisfy these requirements.

As described in Sec. 4.2.1 the operational semantics of a WDEPC model for a business process abstracts away most of the more synthetically specified event nodes between the business function nodes and therefore enables a flexible notion of equivalence, which is based on the special control events and, most importantly, the implicit data flow dependencies between the business functions and their actors. This way, the order of the execution steps is not completely fixed, such that a reordering of them is possible, which enables the generation of several equivalent execution sequences.

In the case that some components are not available, the process execution does not have to immediately stop, because the workflow engine can offer to continue with another equivalent process run, in which the critical step will appear at a later stage, such that the problem can be possibly resolved in between. Furthermore, the workflow engine can choose an alternative execution, which differs only for the subsequent steps, where possibly some continuity snippets are used.

According to Def. 4.2.6 below, two process runs are equivalent, if they are permutations of each other. Furthermore, the definition specifies under which conditions a sequence of steps of the process model is executable with respect to the operational semantics of a WDEPC. In order to improve efficiency of the analysis and the generation of process variants in Sec. 4.3 some security requirements can be transformed into application conditions of the rules of the operational semantics using the construction according to Def. 7.11 in [EEPT06] for graph constraints. This way, some security requirements are checked on-the-fly instead of checking them on the terminal graphs of a derivation only. For this reason, the rules of the operational semantics may contain some negative application conditions (NACs). Hence, the notion of equivalent process runs is already given for rules with NACs. Nevertheless, the complete check of process runs concerning the satisfaction of the func-

tional requirements and the remaining security requirements includes the validation of the corresponding constraints.

Definition 4.2.6 (Equivalent Process Runs)**.** *Given a business process BP modelled as a WDEPC and its continuity snippets CS, and let GG be the derived graph grammar $GG = (TG, SG, P, \pi)$ with induced STS $\mathcal{S}$ defining the operational semantics of BP and CS, where the rules in P are possibly extended by some NACs ensuring additional requirements. A sequence $s = (p_1, \ldots, p_n)$ of rules in the set of rules P is called* executable process run*, if there is a derivation $SG \overset{p_1}{\Longrightarrow} G_1 \Rightarrow \ldots \overset{p_n}{\Longrightarrow} G_n$ in the STS $\mathcal{S}$.*

Two sequences $s_1 = (p_1, \ldots, p_n)$ and $s_2 = (q_1, \ldots, q_m)$ of rules in P are called equivalent process runs*, if they are both executable and s_2 is a permutation of s_1.*

As first main result of this section we show by Thm. 4.2.7 below that for each executable process run the set of its equivalent runs can be obtained using the STS that is induced by the operational semantics of the WDEPC business process model. This means that we can apply the general techniques of Ch. 3 to analyse equivalence of process runs of WDEPC models.

Theorem 4.2.7 (Equivalent Process Runs)**.** *Given a WDEPC business process model, its continuity snippets and the derived graph grammar $GG = (TG, SG, P, \pi)$ specifying the operational semantics of them, where the rules may contain NACs. Let $s_1 = (p_1, \ldots, p_n)$ be an executable process run via derivation $d = (SG \overset{p_1}{\Longrightarrow} G_1 \Rightarrow \ldots \overset{p_n}{\Longrightarrow} G_n)$ and let $\mathcal{S} = STSd$. Then, a sequence s_2 is an equivalent process run of s_1 iff $s_1 \overset{\pi}{\approx}_{\mathcal{S}} s_2$, i.e. if s_2 and s_1 are permutation-equivalent sequences according to Def. 3.4.10.*

Proof. "$\Rightarrow$": Let s_2 be an equivalent process run of s_1 then we have that s_2 is a permutation of s_1 and both sequences induce derivations in $\mathcal{S}$. According to Rem. 3.4.8 switch equivalence without NACs leads to the complete set of equivalent permutations. By Def. 4.2.6 we know that both sequences are executable and therefore, they respect all NACs. By Thm. 3.4.12 and Cor. 3.4.14 we further know that $drv(s_1)$ and $drv(s_2)$ being permutation-equivalent derivations implies that s_1 and s_2 are permutation-equivalent sequences.

"$\Leftarrow$": By Thm. 3.4.12 in Sec. 3.4 we have that two permutation equivalent sequences $s_1 \overset{\pi}{\approx}_{\mathcal{S}} s_2$ in $STS(d)$ specify two permutation-equivalent transformation sequences $drv(s_1) \overset{\pi}{\approx} drv(s_2)$. By Fact 3.4.11 we further have that $drv(s_2)$ is a derivation in $\mathcal{S}$. The rules in the induced STS of GG have possibly additional NACs, which cannot be embedded in the super object T of $STS(d)$. Therefore, none of these additional NACs occurs in an intermediate graph of $drv(s_2)$ and thus, $drv(s_2)$ is also a derivation in the induced STS of GG. □

Example 4.2.8 (Equivalent Process Runs)**.** *There are several equivalent process executions for WDEPC model LG presented in Fig. 4.2 in Sec. 4.1.1. Consider for example the*

case that a client is accepted and all necessary steps are performed. Based on the rules for the operational semantics we can construct the following derivation in the STS that corresponds to the graph grammar GG for the operational semantics.

$$d = (\ SG \xRightarrow{Get\_Customer\_Identity} G_1 \xRightarrow{Get\_Customer\_Demand} G_2 \xRightarrow{Store\_ID\_Data} G_3 \xRightarrow{Store\_CD\_Data} G_4 \xRightarrow{Credit\_Worthiness} G_5 \xRightarrow{Rating\_Customer} G_6 \xRightarrow{Customer\_Acceptance} G_7 \xRightarrow{Create\_Optimized\_Product} G_8 \xRightarrow{Create\_Contract} G_9 \xRightarrow{RM\_Signature} G_{10} \xRightarrow{Customer\_Signature} G_{11} \xRightarrow{Approve\_Contract} G_{12} \xRightarrow{Cash\_Payment} G_{13} \xRightarrow{Cash\_returned\_Loop} G_{14} \xRightarrow{Cash\_returned\_End} G_{15} \xRightarrow{Close\_Contract} G_{16})$$

Based on the derived dependency relations of the STS $STS(d)$ there are several sequences s' of rule names which are permutation-equivalent to $s_1 = seq(d) = \langle 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16\rangle$. Consider, e.g. the following permutation-equivalent sequences:

$$s_2 = \langle 2,1,3,4,5,6,7,8,9,10,11,12,13,14,15,16\rangle$$
$$s_3 = \langle 1,3,2,4,5,6,7,8,9,10,11,12,13,14,15,16\rangle$$
$$s_4 = \langle 1,2,3,4,5,6,7,8,9,11,10,12,13,14,15,16\rangle$$

Sequence s_2 is derived from sequence s_1 by switching the first two independent steps meaning that at first the customer is interviewed for retrieving his demand and thereafter, his personal data is processed. Sequence s_3 is obtained by switching the third and the forth step, which are independent as well, such that the order in which the customer data is typed in the workflow engine can be performed in any order. Finally, sequence s_4 specifies that the customer may sign the contract(11) before the relationship manager has signed it and the relationship manager signs thereafter (10). Since all these sequences of the steps are permutation-equivalent to s_1 within $STS(d)$ we can apply Thm. 4.2.7 and have that the sequences specify equivalent process runs.

In order to additionally ensure the functional and security requirements of the constructed process runs for a given WDEPC business process model we introduce the notion of valid process runs that respect the requirements, which are therefore formalized as graph constraints.

Definition 4.2.9 (Valid Process Runs)**.** *Given a business process BP modelled as a WDEPC and its continuity snippets CS, and let GG be the derived graph grammar $GG = (TG, SG, P)$ with induced STS $\mathcal{S}$ defining the operational semantics of BP and CS. Let CON be a set of graph constraints typed over TG that specify functional and non-functional requirements. A sequence $s = (p_1, \ldots, p_n)$ of rules in the set of rules P is called* valid process run*, if it is executable via a derivation $SG \xRightarrow{p_1} G_1 \Rightarrow \ldots \xRightarrow{p_n} G_n$ in the STS $\mathcal{S}$, such that for each constraint $PC(a) \in CON$ we have that $G_n \models PC(a)$.*

Once the validity of one process run is checked we have by Thm. 4.2.10 below that we can directly derive the equivalent process runs that also respect the functional and security

requirements according to Def. 4.2.9. This means that there is no further need for checking the constraints.

Theorem 4.2.10 (Valid Process Runs)**.** *Given a graph grammar $GG = (TG, SG, P)$ specifying the operational semantics of a WDEPC process model and its continuity snippets with induced STS $\mathcal{S}$. Let CON be a set of graph constraints typed over TG that specify functional and non-functional requirements and let $s_1 = (p_1, \ldots, p_n)$ be a valid process run. Then any permutation-equivalent sequence s_2 of s_1 ($s_1 \overset{\pi}{\approx}_{\mathcal{S}} s_2$) according to Def. 3.4.10 is a valid process run.*

Proof. By Thm. 4.2.7 we have that s_2 is an equivalent process run of s_1 and therefore, s_2 is executable. By permutation equivalence we further have that the derivations $drv(s_1)$ and $drv(s_2)$ are switch-equivalent without NACs and therefore, the terminal graphs are isomorphic and by definition we know that $G_n \models PC(a)$ for each constraint $PC(a) \in CON$. □

The equivalent process runs in Ex. 4.2.8 already satisfy the given functional and security requirements as given in Sec. 4.2.1. In particular, we have that the customer is accepted, the required data elements (CoID, SC, Closing Date) are created and the payment plan is closed. Furthermore, the four-eye principle formalized by the contraint in Fig. 4.9 is ensured, because we have that the relationship manager and the credit officer sign the contract. However, Example 4.2.8 only shows some possible reorderings. The automated and complete generation of the equivalent process runs is presented in the next sections.

4.3 Modification and Optimization

In order to support business continuity management the following section shows how different process executions can be generated from a given WDEPC process model and additional continuity snippets. For this purpose we use the analysis techniques in Sec. 4.2 based on the operational semantics of WDEPC business process models.

On the one hand, the generation of equivalent reorderings allows for optimization with respect to certain criteria, e.g. in order to execute certain business functions in parallel and to reduce some annotated costs. On the other hand, the additional continuity snippets are used to derive complete continuity processes for different combinations of failures. This way, the generation can either purely construct equivalent process runs of the WDEPC process model or, for the more general case, construct also those process runs that use combinations of continuity snippets, which enable executions when some components or actors are not available. Furthermore, the generation technique includes a validation of the functional and security requirements, such that the generated process variants are sound with

respect to these requirements. A summary of the achievements with respect to the problems of business continuity management and the formulated aims is presented in Sec. 4.3.2.

4.3.1 Generation of Valid Process Variants and Continuity Processes

Once a business process model given by a WDEPC and its continuity snippets are modelled they can be analysed using the presented techniques in Sec. 4.2 based on the formal operational semantics of these models. This analysis further enables the construction of concrete valid process runs. Based on such concrete process runs we present in this section how all equivalent and valid ones are generated. For this purpose, the dependency net of the corresponding STS is generated according to Sec. 3.5 using the prototypical tool support described in Ch. 7 and presented in [BHG11]. The generated dependency net purely specifies the dependencies between the process functions and is used to generate the universe of the business process variants and the possible continuity processes.

In order to ensure the security requirements during the generation of the reachability graph the considered example constraint in our case study ensuring the four-eye-principle is transformed into negative application conditions for the rules as described in Sec. 4.2.1 using the concepts in [EEPT06]. The generation ensures that functional as well as security requirements are respected in the way that steps that violate the security requirements are not performed and paths that finally do not fulfil the functional requirements are filtered out. This way, the universe of valid process paths is generated as shown in Fig. 4.11.

The first graph in Fig. 4.11 shows the possible paths of the standard process in Fig. 4.2, where the middle node represents the starting point. Each arrow in the graph represents one step. At first, there are two choices for executing a function, namely, either the relationship manager records the customer identity or the customer demand. In any case the next step can be the retrieval of the other data or the storage of the already recorded data. This flexibility is not directly present in the WDEPC model, but by leaving out the syntactic events that fix the order of the steps, we derive this user-friendly flexibility.

The overall amount of possible sequences of the standard process is 126. All those sequences are *semantically equivalent* with respect to the given functional requirements. The second graph (from left) shows the last 6 functions including the loop at function “F14” leading to 7 steps for each path. Here again, there are two possibilities at the beginning: either the relationship manager signs the contract first or the customer. Usually the relationship manager will sign first, but there might also be few cases in which this is not the case because of time constraints of the customer. Finally, we derive 6 functional valid sequences for this part of the workflow.

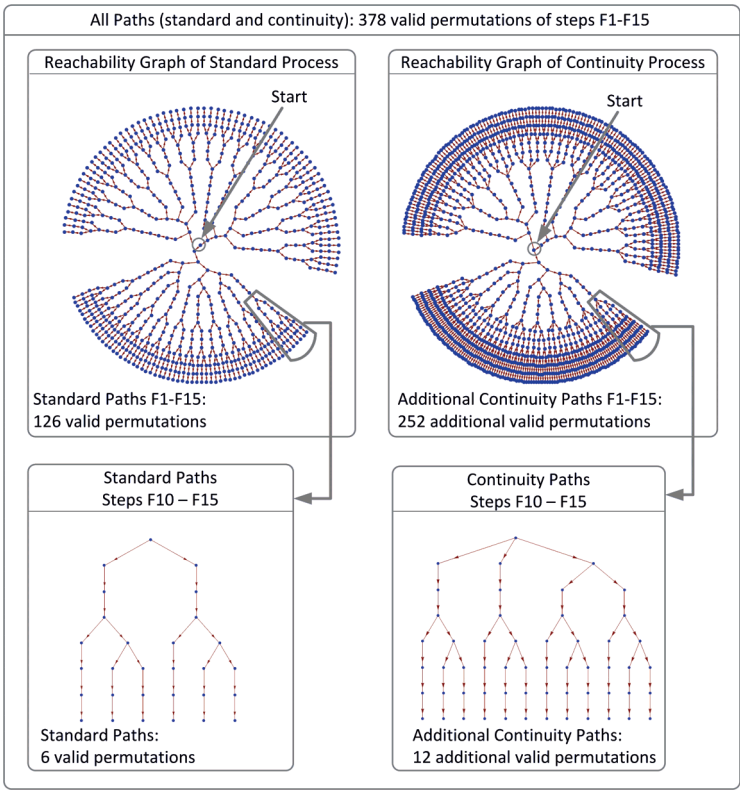


Figure 4.11: Universe of Standard and Continutity Paths

The graphs on the right hand side of Fig. 4.11 show the additional continuity process paths that are possible using one or more continuity snippets in Fig. 4.12 for the business functions “RM Signature (F10)” and “Approve Contract (F12)”.

One or both of the involved actors (RM and CO) at functions F10 (RM Signature) and F12 (Approve Contract) may be unavailable. In these cases, the functions can be replaced using some of the continuity snippets in Fig. 4.12. These functions have to respect the four-eye principle as discussed in Sect. 4.2.1. Indeed, if the snippets F10(C2) and F12(C2) are chosen, then both actors are replaced by the credit advisor. But since the generation checks the security constraint shown in Fig. 4.9 on-the-fly and provides only the valid sequences, this combination is filtered out.

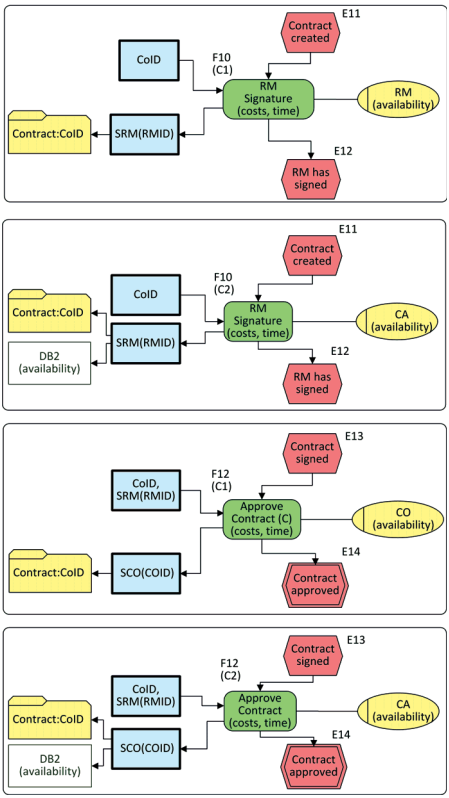


Figure 4.12: Continuity snippets for the functions F10 and F12

The overall amount of additional continuity paths is 252, i.e. there are 252 additional possible process runs that contain at least one continuity snippet. For the last 6 functions of the workflow part there are 12 additional paths as shown by the lower right reachability graph in Fig. 4.11.

4.3.2 Summary of the Solution

Concerning the discussed list in Sec. 4.1.3 of the main problems of business process modelling on which we focussed in this study we now evaluate to which extend the presented techniques for the formal analysis of business process models and for the generation of

equivalent as well as validated standard and continuity runs can be used to achieve the specified goals (see Tab. 4.2 below) for solving the specified problems and improving the modelling process.

Table 4.2: Goals for business process modelling

Criterion	Goal
1. Form of Process Models	Comprehensive, extensive, human-centric visual notation, decreased amount of text documents
2. Form of Requirements	Text documents plus formalized constraints
3. Form of Continuity Plans	compact and concise models
4. Evaluation of Continuity Plans	semi automated, formally verified results
5. Maintenance	improved maintainability, continuity plans are partly updated automatically
6. Consistency	partly automated formal analysis, reduced redundancies

First of all, the introduced extended version of event driven process chains (EPCs), called workflow engine based and data-flow oriented EPCs provides the basis for analysing data flow dependencies in a systematic way. For this purpose we defined a formal operational semantics for WDEPC models, which can be automatically constructed from the underlying abstract syntax. Therefore, the business process models do not have to be specified in a different notation. This way, we meet the goal for criterion 1 (form of process models) as follows. The business process models are human-centric using the common existing visual notation. The presented concept of continuity snippets for modelling alternative process parts for specific failures improves the modelling of continuity plans and facilitates the reduction of text documents that specify complete continuity plans separately for certain failures. Moreover, the continuity snippets can be combined for different possible combinations of possible failures, such that the coverage of the continuity plans can be increased leading to extensive but still comprehensive models.

Concerning criterion 2 (form of requirements) we presented in Sec. 4.2.1 how functional and security requirements can be formalized by graph constraints. Based on the formal operational semantics given by a graph grammar and an induced subobject transformation system the presented formalized functional and security requirements can be automatically verified for the generated process runs as presented in Sec. 4.3.1. Usinge the new notion of continuity snippets we also meet the goal for criterion 3 (form of continuity plans), which demands compact and concise models and the snippets substantially reduce the size of the continuity models. By Thms. 4.2.7 and 4.2.10 the validity of the generated equivalent process runs is formally ensured, which is important for compliance (criterion 4). For

this reason, the generation of equivalent process runs and additional continuity variants provides formally verified results, which ensure the quality of the models.

Since the continuity snippets are kept small the efforts for maintenance (criterion 4) are reduced, because modifications of the standard process usually do not require updates of the snippets, if there if no overlap of business functions. The continuity processes can be automatically updated using the generation techniques. In comparison to maintaining complete continuity process models, the amount of redundancies is substantially reduced, which is required by the goal for criterion 6 (consistency).

Moreover, the presented generation techniques provide several general advantages. In the case of an emergency the management can look at different options and make informed decisions. This way, the technique provides a partly automated recommender service. Furthermore, continuity plans can be checked and ranked regarding their effectiveness and efficiency leading to optimized continuity plans. Depending on the occurring concrete failures, a workflow engine can select an emergency process based on the already computed set of continuity process runs. It can reconfigure the running process instance in order to optimize time and cost functions while respecting the functional and security requirements. Furthermore, the generation allows for fast reactions of the workflow engine towards upcoming failures, which is important for real-time requirements concerning financial transactions. Finally, the business process model and its corresponding continuity snippets can be stored separately, which reduces the complexity of process models and supports ideally the bank's decentralized modelling workflows.

Besides the consistency checks with respect to the functional and security constraints we show in Ch. 6 how the business process models are checked for conformance to their corresponding business service structure models using the model transformation techniques of Ch. 5. Moreover, we presented in [BHG11] how the process models can be analysed for information-flow properties, like partial distribution of confidential data based on analysis techniques and tool support for the $mCRL2$ process algebra.

Summing up, the presented results and techniques substantially support the overall modelling process and provide important contributions concerning the specified goals for business process modelling including business continuity management.

Chapter 5

Model Transformation Based on Triple Graph Grammars

Model transformations based on triple graph grammars (TGGs) have been introduced by Schürr [Sch94] and are used for the specification and execution of bidirectional model transformations between domain specific languages (DSLs). The power of bidirectional model transformations is based on the simultaneous support of transformations in both forward and backward direction. This way, models can be maintained in two repositories – one for diagrams in the source domain and one for diagrams in the target domain. The modellers can work in separated teams, and the specified model transformations are used to support the interchange between these groups and their models. In particular, a modeller can generate models in one domain from models in another domain using the concepts for model transformation in Sec. 5.1 and he can additionally validate and ensure syntactical correctness and completeness using the results and analysis techniques in Sec. 5.2.

In addition to the general advantages of bidirectional model transformations, TGGs simplify the design of model transformations. A single set of triple rules is sufficient to generate the operational rules for the forward and backward model transformations. Furthermore, model transformations based on TGGs preserve a given source model by creating a separate target model together with a correspondence structure. This way, the given models are not modified, which is especially important for data base driven model repositories. Moreover, TGGs specify model transformations based on the abstract syntax of DSLs and are therefore not restricted to specific types of modelling languages.

The first main contribution of this chapter shows the main challenges for model transformations as well as main concepts and constructions for model transformations based on TGGs. Apart from model transformations based on forward rules (Sec. 5.1.1), including an on-the-fly construction, we additionally present model transformations based on forward translation rules (Sec. 5.1.2), where we improved further the execution of model transformations by additional translation attributes. As the first main result we

show the equivalence of both approaches in Thm. 5.1.34. Sec. 5.2 presents the second main contribution concerning the analysis of model transformations, where we show syntactical correctness, completeness and termination of model transformations based on TGGs (Thm. 5.2.1, Thm. 5.2.2, Thm. 5.2.4 and Cor. 5.2.6). Moreover, we provide powerful results for analysing functional behaviour of model transformations (Thms. 5.2.27 and 5.2.31) and information preservation (Thms. 5.2.34 and 5.2.37). Finally, in Sec. 5.3 we present techniques for improving a model transformation with respect to efficiency based on the previous results, techniques for the detection and reduction of unintended behaviour (Thm. 5.3.7), and we provide a benchmark that validates the power of the approach.

5.1 Concepts and Characteristics

Model transformations appear in several contexts, e.g. in the various facets of model driven architecture [Gro09] encompassing model refinement and interoperability of system components. The involved languages can be closely related or they can be more heterogeneous, e.g. in the special case of model refactoring, the source language and the target language are the same. From a general point of view, a model transformation $MT : VL_S \Rightarrow VL_T$ between visual languages transforms models from the source language VL_S to models of the target language VL_T. Main challenges were described in [SK08] for model transformation approaches based on triple graph grammars. In [HHK10] we extended this list and also the scope, and we described general challenges for model transformations. The extended list of challenges is given below divided into two dimensions. In the subsequent sections of this chapter we present main results for TGGs concerning these challenges.

There are two dimensions which contain major challenges for model transformations, concerning on the one hand functional aspects, and on the other hand non-functional aspects. The first dimension, called *functional dimension*, concerns the reliability of the produced results. Depending on the concrete application of a model transformation $MT : VL_S \Rightarrow VL_T$, some of the following properties may have to be ensured.

1. *Syntactical Correctness:* For each model $M_S \in VL_S$ that is transformed by MT the resulting model M_T has to be syntactically correct, i.e. $M_T \in VL_T$.

2. *Semantic Correctness:* The semantics of each model $M_S \in VL_S$ that is transformed by MT has to be preserved or reflected, respectively.

3. *Completeness:* The model transformation MT can be performed on each model $M_S \in VL_S$. Additionally, it can be required that MT reaches all models $M_T \in VL_T$.

4. *Functional Behaviour:* For each source model M_S, the model transformation MT will always terminate and lead to the same resulting target model M_T.

The second dimension, called *non-functional dimension*, concerns usability and applicability aspects of model transformations. Therefore, some of the following challenges are also main requirements.

1. *Efficiency:* Model transformations should have polynomial space and time complexity. Furthermore, there may be further time constraints that need to be respected, depending on the application domain and the intended way of use.

2. *Intuitive Specification:* The specification of model transformations can be performed based on patterns that describe how model fragments in a source model correspond to model fragments in a target model. If the source (resp. target) language is a visual language, then the components of the model transformation can be visualized using the concrete syntax of the visual language.

3. *Maintainability:* Extensions and modifications of a model transformation should only require little efforts. Side effects of local changes should be handled and analysed automatically.

4. *Expressiveness:* Special control conditions and constructs have to be available in order to handle more complex models, which, e.g., contain substructures with a partial ordering or hierarchies.

5. *Bidirectional model transformations:* The specification of a model transformation should provide the basis for both a model transformation from the source to the target language and a model transformation in the inverse direction.

In the following sections we present suitable techniques for the specification of model transformations based on TGGs. These techniques provide validated and verified capabilities for a wide range of the challenges listed above.

5.1.1 Model Transformation Based on Forward Rules

Triple graph grammars [Sch94] are a well known approach for bidirectional model transformations. Models are defined as pairs of source and target graphs, which are connected via a correspondence graph and its embeddings into the source and target graphs. In [KS06], Königs and Schürr formalize the basic concepts of triple graph grammars in a set-theoretical way, which is generalized and extended in [EEE+07] to typed, attributed graphs. In this section, we review the corresponding main constructions for model transformations based on triple graph grammars with further extensions according to [EHS09a, EEHP09b].

Triple graphs combine source and target models via a correspondence structure given by a correspondence graph and two morphisms linking the correspondence graph with the underlying graphs, which specify the abstract syntax of the source and target models.

Definition 5.1.1 (Triple Graph)**.** *A* triple graph $G = (G^S \xleftarrow{s_G} G^C \xrightarrow{t_G} G^T)$ *consists of three graphs* G^S, G^C, *and* G^T, *called source, correspondence, and target graphs, together with two graph morphisms* $s_G : G^C \to G^S$ *and* $t_G : G^C \to G^T$.

Throughout this chapter we use a common case study for model transformations from UML Class Diagrams [OMG07] to relational data base models. In order to keep our examples simple but expressive we use simplified diagrams, which contain the main concepts of both languages. In detail, class diagram models use the concepts inheritance, associations and attributes and relational data base models use primary as well as foreign keys. Furthermore, we use attribution within the abstract syntax, such that an extension to further concepts of the DSLs is supported.

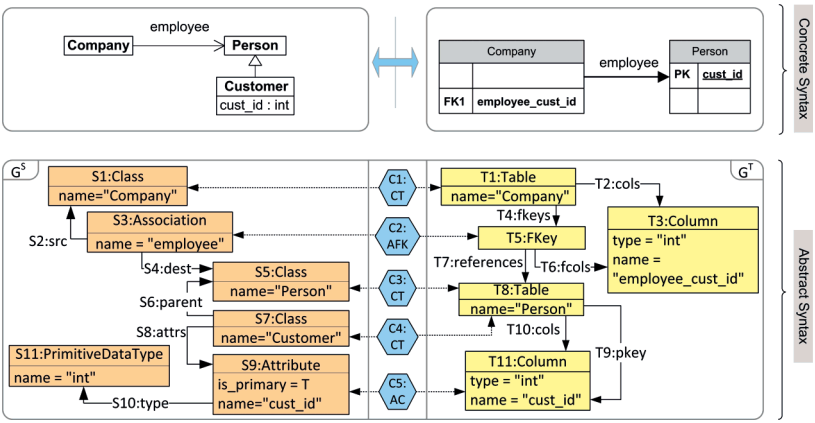


Figure 5.1: Triple graph instance for *CD2RDBM*

Example 5.1.2 (Triple Graph)**.** *Fig. 5.1 shows a triple graph which specifies a class diagram in its source and a data base model in its target component. The upper line shows both diagrams in concrete syntax and the lower part shows them in abstract syntax based on typed attributed graphs [EEPT06]. The example contains all relevant concepts that we are going to handle within this case study. The first view shows that classes are going to be translated to tables, attributes to columns and associations to foreign keys. Since data base models in the plain form do not provide inheritance information, we relate subclasses (e.g. the node S7 with name "Customer") to the table that is connected to the super class already. We will come back to this example triple graph at the end of this section and see in detail how the models are transformed into each other. But first, we introduce the main formal concepts for this purpose.*

In order to define typed triple graphs and triple rules we first define triple graph morphisms, such that we can set up the category of triple graphs **TripleGraphs**. A triple graph morphism $m = (m^S, m^C, m^T) : G \to H$ consists of three graph morphisms $m^S : G^S \to H^S$, $m^C : G^C \to H^C$ and $m^T : G^T \to H^T$ such that $m^S \circ s_G = s_H \circ m^C$ and $m^T \circ t_G = t_H \circ m^C$. It is called injective if all components are injective. A typed triple graph $(G, type_G)$ consists of a triple graph G and a triple graph morphism $type_G : G \to TG$ for the typing over TG.

Definition 5.1.3 (Category of Triple Graphs)**.** *The category* **TripleGraphs** *consists of triple graphs and triple graph morphisms. Analogously, category* **TripleAGraphs** *of attributed triple graphs is a diagram category over the category of attributed graphs* **AGraphs***(cf. Def. 3.1.5 in Sec. 3.1). Moreover, given type graphs TG in* **TripleGraphs** *(resp. ATG in* **TripleAGraphs***) we obtain category* $\mathbf{TripleGraphs}_{TG}$ *consisting of typed triple graphs (resp.* $\mathbf{TripleAGraphs}_{ATG}$ *consisting of typed attributed triple graphs) as slice categories over* **TripleGraphs** *(resp.* **TripleAGraphs***).*

In [EEE+07] we have shown that **TripleGraphs** is an $\mathcal{M}$-adhesive category for the class $\mathcal{M}$ of all injective triple graph morphisms. Accordingly, the categories (**TripleAGraphs**,$\mathcal{M}$), ($\mathbf{TripleGraphs}_{TG}, \mathcal{M}$), ($\mathbf{TripleAGraphs}_{ATG}, \mathcal{M}$) are shown to be $\mathcal{M}$-adhesive, where $\mathcal{M}$ is the class of morphisms that are injective on the graph part and isomorphic on the data part for the case with attribution. Therefore, we can apply the results shown for $\mathcal{M}$-adhesive categories in general. In particular, the Local-Church-Rosser and Concurrency Thms. are used for showing the fundamental composition and decomposition theorem for TGGs (Thm. 5.1.14).

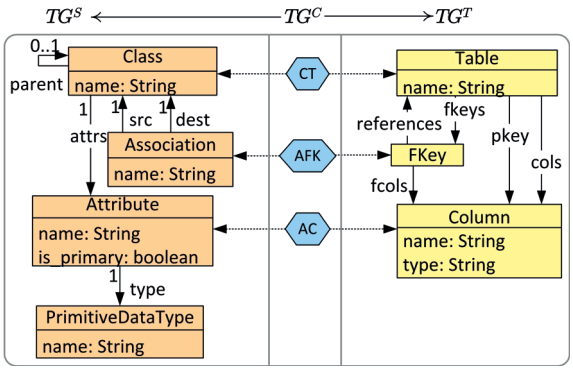


Figure 5.2: Triple type graph for *CD2RDBM*

Example 5.1.4 (Triple Type Graph)**.** *Fig. 5.2 shows the type graph TG of the triple graph grammar TGG for our example model transformation* CD2RDBM *from class diagrams to database models. The source component TG^S defines the structure of class diagrams while in its target component the structure of relational database models is specified. Classes correspond to tables, attributes to columns, and associations to foreign keys. Throughout the example, originating from [EEE+07], elements are arranged left, center, and right according to the component types source, correspondence and target. Morphisms starting at a correspondence part are specified by dashed arrows. Furthermore, the triple rules of the grammar shown in Figures 5.4-5.5 ensure several multiplicity constraints, which are denoted within the type graph. In addition, the source language only contains class diagrams where classes have unique primary attributes.*

Note that the case study uses attributed triple graphs based on E-graphs as presented in [EEE+07] in the framework of weak adhesive HLR which are special $\mathcal{M}$-adhesive categories (see Def. 3.1.11).

Triple rules of a triple graph grammar TGG – called TGG-triple rules – specify how models in the source and target DSL can be synchronously created including the correspondence structures that relate them. This implies that TGG-triple rules are non-deleting, while triple rules for other purposes may be also deleting as e.g. the generated forward translation rules in Sec. 5.1.2, where we extend the notion of triple rules to the deleting case.

Definition 5.1.5 (Triple Rule and Triple Graph Transformation Step)**.** *A* triple rule tr *is an injective triple graph morphism $tr = (tr^S, tr^C, tr^T) : L \to R$ and w.l.o.g. we assume tr to be an inclusion. Given a triple graph morphism $m : L \to G$, a triple graph transformation step (*TGT*-step) $G \overset{tr,m}{\Longrightarrow} H$ from G to a triple graph H via tr is given by a pushout in* **TripleGraphs** *with comatch $n : R \to H$ and transformation inclusion $t : G \hookrightarrow H$.*

$$\begin{array}{ccccc} L = (L^S & \xleftarrow{s_L} & L^C & \xrightarrow{t_L} & L^T) \\ {\scriptstyle tr}\downarrow & {\scriptstyle tr^S}\downarrow & & {\scriptstyle tr^C}\downarrow & {\scriptstyle tr^T}\downarrow \\ R = (R^S & \xleftarrow[s_R]{} & R^C & \xrightarrow[t_R]{} & R^T) \end{array} \qquad \begin{array}{ccc} L & \overset{tr}{\hookrightarrow} & R \\ {\scriptstyle m}\downarrow & (PO) & \downarrow{\scriptstyle n} \\ G & \underset{t}{\hookrightarrow} & H \end{array}$$

A sequence of TGT*-steps $G_0 \overset{tr_1,m_1}{\Longrightarrow} G_1 \overset{tr_2,m_2}{\Longrightarrow} \ldots \overset{tr_n,m_n}{\Longrightarrow} G_n$ is called* TGT*-sequence and we write $G_0 \overset{tr^*}{\Longrightarrow} G_n$ for short. The trace of a* TGT*-sequence is given by the composition of the transformation inclusions, i.e. $trace(G_0) = id_{G_0}$ and $trace(G_0 \overset{tr^*}{\Longrightarrow} G_n \overset{tr_{n+1},m_{n+1}}{\Longrightarrow} G_{n+1}) = t_{n+1} \circ trace(G_0 \overset{tr^*}{\Longrightarrow} G_n)$. Moreover, given a subsequence $G_i \overset{tr'^*}{\Longrightarrow} G_j$ of a* TGT*-sequence $G_0 \overset{tr^*}{\Longrightarrow} G_n$ leaving out the first i steps and the last $n - j$ steps, then the corresponding subtrace is defined by $trace(G_0 \overset{tr^*}{\Longrightarrow} G_n)_{i,j} = trace(G_i \overset{tr'^*}{\Longrightarrow} G_j)$.*

According to [EEE+07] a pushout in **TripleGraphs** is constructed componentwise in **Graphs**and the morphisms s_H and t_H of triple graph H are induced by the pushout in the correspondence component as shown in Fig. 5.3 below.

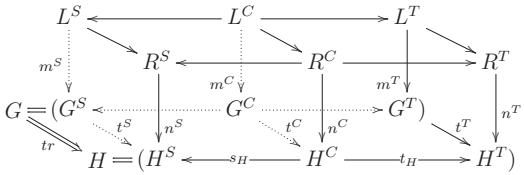


Figure 5.3: Triple Transformation Step in **Graphs**

Definition 5.1.6. *A triple graph grammar $TGG = (TG, SG, TR)$ consists of a triple graph TG, called* triple type graph*, a triple graph SG, called* triple start graph*, and a set TR of triple rules.*

We require the start graphs of TGGs to be empty, i.e. $SG = \varnothing$, which is common practice (see [Sch94, SK08, EEE+07]) and necessary already for the fundamental composition and decomposition result in Thm. 5.1.14 at the end of this section. The effect of a non-empty start graph can be simulated by an additional TGG-triple rule $tr : \varnothing \to SG$ also called axiom (cf. [SK08]). However, if explicit non-empty start graphs become of interest in future applications the results for TGGs will have to be extended to this case.

The language of integrated models VL is generated by all possible transformation sequences via the triple rules of a TGG. In order to formalize the domain and codomain of correct model transformation sequences we define the sets VL_S of source and VL_T of target models by a restriction of the integrated models to the source and target components, respectively.

Definition 5.1.7 (Triple, Source and Target Language)**.** *A set of TGG-triple rules TR defines the* triple language $VL = \{G \mid \varnothing \Rightarrow^* G \text{ via } TR\}$ *of triple graphs.* Source language VL_S *and* target language *are derived by projection to the triple components, i.e. $VL_S = proj_S(VL)$ and $VL_T = proj_T(VL)$, where $proj_X$ is a projection defined by restriction to one of the triple components, i.e. $X \in \{S, T\}$.*

Example 5.1.8 (Triple Rules)**.** *The triple rules in Fig. 5.4 are part of the rules of the grammar TGG for the model transformation $CD2RDBM$. They are presented in short notation, i.e. left and right hand side of a rule are depicted in one triple graph. Elements which are created by the rule are labelled with green "++" and marked by green line colouring. The rule "*Class2Table*" synchronously creates a class with name "n" together*

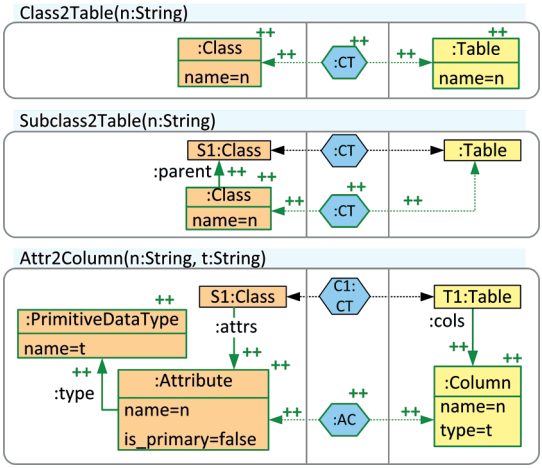


Figure 5.4: Rules for the model transformation *CD2RDBM*, Part 1

*with the corresponding table in the relational database. Accordingly, subclasses are connected to the tables of its super classes by rule "*Subclass2Table*". Attributes with type "t" are created together with their corresponding columns in the database component via the rule "*Attr2Column*".*

According to [EHS09a, EHS09b, EEHP09b], we present negative application conditions for triple rules. In most case studies of model transformations source-target NACs, i.e. either source or target NACs, are sufficient and we regard them as the standard case. They prohibit the existence of certain structures either in the source or in the target part only, while general NACs may prohibit both at once.

Definition 5.1.9 (Triple Rule with Negative Application Conditions)**.** *Given a triple rule* $tr = (L \to R)$*, a negative application condition (NAC)* $(n : L \to N)$ *consists of a triple graph* N *and a triple graph morphism* n*. A NAC with* $n = (n^S, id_{L^C}, id^{L_T})$ *is called* source NAC *and a NAC with* $n = (id_{L^S}, id_{L^C}, n^T)$ *is called* target NAC*.*

A match $m : L \to G$ *is NAC consistent if there is no injective* $q : N \to G$ *such that* $q \circ n = m$ *for each NAC* $L \xrightarrow{n} N$*. A triple transformation* $G \overset{*}{\Rightarrow} H$ *is NAC consistent if all matches are NAC consistent.*

Example 5.1.10 (Triple Rules with NACs)**.** *Figure 5.5 shows the remaining two triple rules "*$Association2ForeignKey$*" and "*$PrimaryAttr2Column$*" for the model transformation "*$CD2RDBM$*". NACs are specified in short notation using the label "NAC" with*

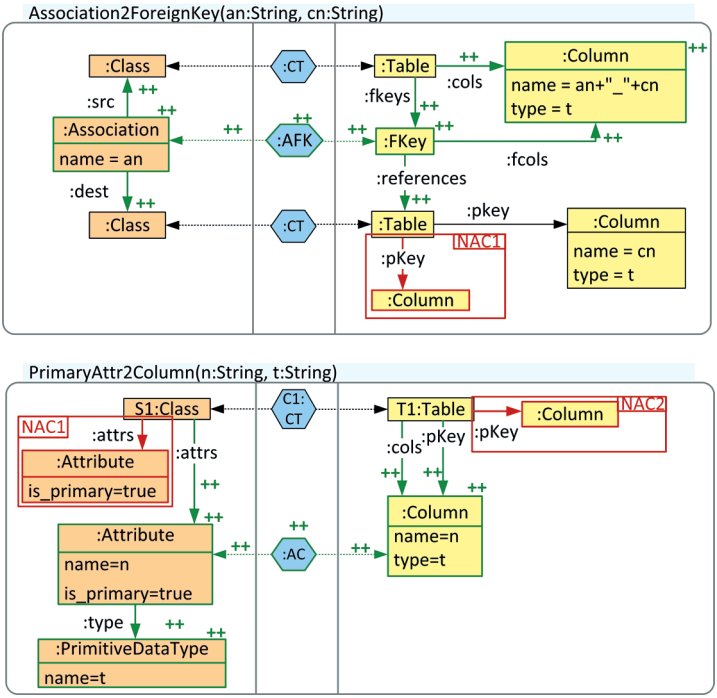


Figure 5.5: Rules for the model transformation *CD2RDBM*, Part 2

a frame and red line colour within the frame. A complete NAC is obtained by composing the left hand side of a rule with the red marked elements within the NAC-frame. The rule "Association2ForeignKey" creates an association between two classes and the corresponding foreign key and the NAC ensures that there is only one primary key at the destination table. The parameters "an" and "cn" are used to set the names of the association and column nodes. The rule "PrimaryAttr2Column" extends "Attr2Column" by creating additionally a link of type "pkey" for the column and by setting "is_primary=true". Furthermore, there is a source and a target NAC, which ensure that there is neither a primary attribute nor a column currently present.

From each triple rule tr we derive a source rule tr_S for the construction resp. parsing of a model of the source language and a forward rule tr_F for forward transformation sequences (see Fig. 5.6). By TR_S and TR_F we denote the sets of all source and forward rules derived from the set of triple rules TR. Analogously, we derive a target rule tr_T and a backward

rule tr_B for the construction and transformation of a model of the target language leading to the sets TR_T and TR_B.

$$\begin{array}{ccccc} L = (L^S & \xleftarrow{s_L} & L^C & \xrightarrow{t_L} & L^T) \\ {\scriptstyle tr}\downarrow \quad {\scriptstyle tr^S}\downarrow & & {\scriptstyle tr^C}\downarrow & & \downarrow{\scriptstyle tr^T} \\ R = (R^S & \xleftarrow[s_R]{} & R^C & \xrightarrow[t_R]{} & R^T) \end{array}$$

triple rule tr

$$\begin{array}{ccccc} L_S = (L^S & \longleftarrow & \varnothing & \longrightarrow & \varnothing) \\ {\scriptstyle tr_S}\downarrow \quad {\scriptstyle tr^S}\downarrow & & \downarrow & & \downarrow \\ R_S = (R^S & \longleftarrow & \varnothing & \longrightarrow & \varnothing) \end{array}$$

source rule tr_S

$$\begin{array}{ccccc} L_T = (\varnothing & \longleftarrow & \varnothing & \longrightarrow & L^T) \\ {\scriptstyle tr_T}\downarrow & & \downarrow & & {\scriptstyle tr^T}\downarrow \\ R_T = (\varnothing & \longleftarrow & \varnothing & \longrightarrow & R^T) \end{array}$$

target rule tr_T

$$\begin{array}{ccccc} L_F = (R^S & \xleftarrow{tr^S \circ s_L} & L^C & \xrightarrow{t_L} & L^T) \\ {\scriptstyle tr_F}\downarrow \quad {\scriptstyle id}\downarrow & & {\scriptstyle tr^C}\downarrow & & \downarrow{\scriptstyle tr^T} \\ R_F = (R^S & \xleftarrow[s_R]{} & R^C & \xrightarrow[t_R]{} & R^T) \end{array}$$

forward rule tr_F

$$\begin{array}{ccccc} L_B = (L^S & \xleftarrow{s_L} & L^C & \xrightarrow{tr^T \circ t_L} & R^T) \\ {\scriptstyle tr_B}\downarrow \quad {\scriptstyle tr^S}\downarrow & & {\scriptstyle tr^C}\downarrow & & \downarrow{\scriptstyle id} \\ R_B = (R^S & \xleftarrow[s_R]{} & R^C & \xrightarrow[t_R]{} & R^T) \end{array}$$

backward rule tr_B

$$\begin{array}{ccccc} L_{ST} = (L^S & \longleftarrow & \varnothing & \longrightarrow & L^T) \\ {\scriptstyle tr_{ST}}\downarrow \quad {\scriptstyle tr^S}\downarrow & & \downarrow & & \downarrow{\scriptstyle tr^T} \\ R_{ST} = (R^S & \longleftarrow & \varnothing & \longrightarrow & R^T) \end{array}$$

source-target rule tr_{ST}

$$\begin{array}{ccccc} L_I = (R^S & \xleftarrow{tr^S \circ s_L} & L^C & \xrightarrow{tr^T \circ t_L} & R^T) \\ {\scriptstyle tr_I}\downarrow \quad {\scriptstyle id}\downarrow & & {\scriptstyle tr^C}\downarrow & & \downarrow{\scriptstyle id} \\ R_I = (R^S & \xleftarrow[s_R]{} & R^C & \xrightarrow[t_R]{} & R^T) \end{array}$$

integration rule tr_I

operational rule	derived NACs
source rule $tr_S = (tr^S, \varnothing, \varnothing)$	target NACs are removed and each source NAC $n = (n^S, id, id) : L \to (N^S \xleftarrow{s_N} L^C \to L^T)$ is replaced by $n_S = (n^S, \varnothing, \varnothing) : L_S \to (L^S \leftarrow \varnothing \to \varnothing)$
target rule $tr_T = (\varnothing, \varnothing, tr^T)$	source NACs are removed and each target NAC $n = (id, id, n^T) : L \to (L^S \leftarrow L^C \xrightarrow{t_N} N^T)$ is replaced by $n_T = (\varnothing, \varnothing, n^T) : L_T \to (\varnothing \leftarrow \varnothing \to N^T)$
source-target rule $tr_{ST} = (tr^S, \varnothing, tr^T)$	each source-target NAC $n = (n^S, id, n^T) : L \to (N^S \xleftarrow{s_N} L^C \xrightarrow{t_N} N^T)$ is replaced by $n_{ST} = (n^S, \varnothing, n^T) : L_{ST} \to (N^S \leftarrow \varnothing \to N^T)$
forward rule $tr_F = (id, tr^C, tr^T)$	source NACs are removed and each target NAC $n = (id, id, n^T) : L \to (L^S \xleftarrow{s_N} L^C \to N^T)$ is replaced by $n_F = (id, id, n^T) : L_F \to (R^S \xleftarrow{tr^S \circ s_N} N^C \to N^T)$
backward rule $tr_B = (tr^S, tr^C, id)$	target NACs are removed and each source NAC $n = (n^S, id, id) : L \to (N^S \leftarrow L^C \xrightarrow{t_N} L^T)$ is replaced by $n_B = (n^T, id, id) : L_B \to (N^S \leftarrow L^C \xrightarrow{tr^T \circ t_N} R^T)$
integration rule $tr_I = (id, tr^C, id)$	all NACs are removed

Figure 5.6: Derived operational rules of a TGG-triple rule

Definition 5.1.11 (Derived Operational Rules)**.** *Given a TGG-triple rule* $tr = (tr^S, tr^C, tr^T) : L \to R$ *with source-target NACs the source* (tr_S)*, target* (tr_T)*, forward* (tr_F)*, backward* (tr_B) *as well as source-target* (tr_{ST}) *and integration rules* (tr_I) *are derived according to Fig. 5.6, which shows how the rules are derived by taking* tr *and redefining some of their components including their NACs.*

Note that a source rule tr_S may be applicable to triple graphs G even if the corresponding triple rule tr is not applicable, because the left hand side of tr_S in the C- and T- component is smaller or equal. For this reason, the set VL_{S0} of models that can be generated resp. parsed by the set of all source rules TR_S is possibly larger than VL_S in Def. 5.1.7 and we have $VL_S \subseteq VL_{S0} = \{G^S \mid \varnothing \Rightarrow^* (G^S \leftarrow \varnothing \rightarrow \varnothing)\ via\ TR_S\}$. Analogously, we have $VL_T \subseteq VL_{T0} = \{G^T \mid \varnothing \Rightarrow^* (G^T \leftarrow \varnothing \rightarrow \varnothing)\ via\ TR_T\}$.

In the general case, the DSL $\mathcal{L}_S$ for the source domain is specified separately from the TGG for the model transformation. The main approaches for the specification of DSLs are on the one hand meta modelling based on a meta model and a set of well-formedness constraints (see the MOF-Approach [MOF06]) and on the other hand graph grammars consisting of a type graph, which corresponds to the meta model, and a set of transformation rules for the generation of the language (see e.g. the Henshin Tool Suite based on EMF-transformations [ABJ+10]). For this reason, we do not require that $\mathcal{L}_S$ is equal to either VL_{S0} or VL_S. It is sufficient that there is an inclusion $\mathcal{L}_S \subseteq VL_{S0}$ implying that the model transformation is capable to handle any source model $G^S \in \mathcal{L}_S$. This separation of the specification of the source language $\mathcal{L}_S$ and the specification of the TGG for the model transformation additionally enables to reduce the complexity of the developed TGG. The TGG-triple rules do not need to guarantee certain language constraints of the source DSL, because we can require that the given source model is already well-formed with respect to the source language.

In order to perform model transformations based on the derived forward rules the application of the rules has to be controlled in a suitable way. As we have shown in [EEE+07, EHS09a] source consistency is a suitable characterization of a control condition that ensures both, correctness and completeness, which we present in detail in Sec. 5.2.1 concerning the analysis of model transformations. Intuitively, source consistency ensures that the elements in the source model are translated exactly once and furthermore, that the derived target model together with the given source model are related, such that they belong to the language of integrated models $VL = \{G \mid \varnothing \overset{tr^*}{\Longrightarrow} G \text{ via } TR\}$ generated by the set of triple rules TR of the TGG. Thus a model transformation based on forward rules $MT_F : VL_S \Rrightarrow VL_T$ from a visual source language VL_S to a target language VL_T consist of the model transformation sequences $(G^S, G_0 \overset{tr_F^*}{\Longrightarrow} G_n, G^T)$, where $G^S \in VL_S$ is the given source model, $G_0 \overset{tr_F^*}{\Longrightarrow} G_n$ is a source consistent forward transformation sequence

with $G_0 = (G^S \leftarrow \varnothing \rightarrow \varnothing)$, $G_n = (G^S \leftarrow G^C \rightarrow G^T)$, tr_F^* is a sequence of applied forward rules and G^T is the resulting target model.

Definition 5.1.12 (Match and Source Consistency)**.** *Let tr_S^* and tr_F^* be sequences of source rules $tr_{i,S}$ and forward rules $tr_{i,F}$, which are derived from the same TGG-triple rules tr_i for $i = 1, \ldots, n$. Let further $G_{00} \overset{tr_S^*}{\Longrightarrow} G_{n0} \overset{tr_F^*}{\Longrightarrow} G_{nn}$ be a* TGT*-sequence with $(m_{i,S}, n_{i,S})$ being match and comatch of $tr_{i,S}$ (respectively $(m_{i,F}, n_{i,F})$ for $tr_{i,F}$) then* match consistency *of $G_{00} \overset{tr_S^*}{\Longrightarrow} G_{n0} \overset{tr_F^*}{\Longrightarrow} G_{nn}$ means that the S-component of the match m_i is uniquely determined by the comatch $n_{i,S}$ $(i = 1, \ldots, n)$. More precisely, we have that $m_{i,F}^S = trace(G_{00} \overset{tr_S^*}{\Longrightarrow} G_{n0})_{i,n}^S \circ n_{i,S}^S = t_n^S \circ \cdots \circ t_i^S \circ n_{i,S}^S$.*
A TGT*-sequence $G_{n0} \overset{tr_F^*}{\Longrightarrow} G_{nn}$ is* source consistent*, if there is a match consistent sequence $\emptyset \overset{tr_S^*}{\Longrightarrow} G_{n0} \overset{tr_F^*}{\Longrightarrow} G_{nn}$.*

Note that by source consistency the application of the forward rules is controlled by the source sequence, which generates the given source model. Furthermore, if a forward sequence is source consistent then the corresponding source sequence is also unique, because the matches and comatches of the source sequence are uniquely derived from the matches of the forward sequence. The notion of source consistency builds the basis for the following composition and decomposition result which shows the one-to-one correspondence between TGT-sequences based on TGG-triple rules and source consistent forward sequences. Each TGT-sequence based on TGG-triple rules can be decomposed into a source and a forward sequence that are match consistent, which means that the forward sequence is source consistent. Vice versa, each source consistent forward sequence $G_0 \overset{tr_F^*}{\Longrightarrow} G_n$ leads to a source sequence and a forward sequence which are match consistent and can be composed to the corresponding TGT-sequence based on TGG-triple rules.

Remark 5.1.13 (General Assumption for Model Transoformations)**.** *In general, we consider a triple graph grammar $TGG = (TG, SG, TR)$ having an empty start graph $SG = \varnothing$ and triple rules TR with source and target NACs and derived operational source rules TR_S and forward rules TR_F.*

Theorem 5.1.14 (Composition and Decomposition of TGT-Sequences with NACs)**.**

1. **Decomposition***: For each TGT-sequence $G_0 \overset{tr_1}{\Longrightarrow} G_1 \Rightarrow \ldots \overset{tr_n}{\Longrightarrow} G_n$ (1) with NACs there is a corresponding match consistent TGT-sequence $G_0 = G_{00} \overset{tr_{1S}}{\Longrightarrow} G_{10} \Rightarrow \ldots \overset{tr_{nS}}{\Longrightarrow} G_{n0} \overset{tr_{1F}}{\Longrightarrow} G_{n1} \Rightarrow \ldots \overset{tr_{nF}}{\Longrightarrow} G_{nn} = G_n$ (2) with NACs.*

2. **Composition:** *For each match consistent transformation sequence (2) with NACs there is a canonical transformation sequence (1) with NACs.*

3. **Bijective Correspondence:** *Composition and decomposition are inverse to each other (up to isomorphism).*

Proof (Idea). The proof is based on the Concurrency and the Local Church Rosser Thms. for **TripleGraphs** and shown in given in [EEE+07] for the case without NACs, extended to the case with NACs in [EHS09a] for injective matching and extended to general matching and more general application conditions in [GEH11, Gol10]. Given a TGT-sequence (1), then each step $G_{i-1} \overset{tr_i,m_i}{\Longrightarrow} G_i$ can be split into a sequence $G_{i-1} \overset{tr_{i,S},m_{i,S}}{\Longrightarrow} H \overset{tr_{i,F},m_{i,F}}{\Longrightarrow} G_i$ by the Concurrency Thm. with possibly some dependency on the source component. Furthermore, each derived source step $G_{i-1} \overset{tr_{i,S},m_{i,S}}{\Longrightarrow} H$ is sequentially independent from all forward steps $H' \overset{tr_{j,F},m_{j,F}}{\Longrightarrow} G_j$ with $j < i$. This allows by the local Church-Rosser Thm. to switch the steps such that all source steps are moved infront of the forward steps leading to a match consistent sequence (2). Vice versa, given a match-consistent sequence (2) we can reorder the sequentially independent steps as before and we can construct the concurrent TGT-steps pairwise leading to sequence (1). □

The additionally derived source-target rules and integration rules are used for model integration, i.e. the integration of a pair of models, one of the source and one of the target domain. Model integration is not in the main focus of this thesis; however model transformation and integration are closely related. In [EEH08a, EEH08b] we show a similar composition and decomposition result for source-target and integration sequences using the notions "S-T"-match consistency and "source-target consistency".

Based on the concept of source consistency we now define model transformations based on forward rules that are derived from the TGG-triple rules of a triple graph grammar. Thereafter we present an on-the-fly construction for executing these model transformations.

Definition 5.1.15 (Model Transformation based on Forward Rules)**.** *A* model transformation sequence $(G^S, G_0 \overset{tr_F^*}{\Longrightarrow} G_n, G^T)$ based on forward rules *consists of a source graph G^S, a target graph G^T, and a source consistent forward* TGT*-sequence $G_0 \overset{tr_F^*}{\Longrightarrow} G_n$ with $G^S = G_0^S$ and $G^T = G_n^T$.*
A model transformation $MT : VL_{S0} \Rrightarrow VL_{T0}$ based on forward rules *is defined by all model transformation sequences $(G^S, G_0 \overset{tr_F^*}{\Longrightarrow} G_n, G^T)$ with $G^S \in VL_{S0}$ and $G^T \in VL_{T0}$. All the corresponding pairs (G^S, G^T) define the* model transformation relation $MTR_F \subseteq VL_{S0} \times VL_{T0}$ *based on forward rules. Moreover, the corresponding* backward model transformation *is defined analogously, where "source" is replaced by "target", "forward" by "backward" and we have in particular $G^T = G_0^T$ and $G^S = G_n^S$.*

A model transformation based on forward rules can be executed using the on-the-fly construction which we introduced in [EEHP09b]. Compared to a direct construction of the

possible source and forward sequences, which are then checked to be source consistent, the on-the-fly construction checks source consistency during the construction, such that misleading sequences can be neglected at early stages leading to fewer cases for backtracking. The main idea is to synchronously build up the source and the forward sequences and to check for partial source consistency, i.e. for the current comatch of the source step and its corresponding match of the forward step. For this purpose, we define partial match consistency of the forward step leading to the notion of partial source consistency for the current forward sequence.

Definition 5.1.16 (Partial Match Consistency)**.** *Let TR be a set of triple rules with source and target $NACs$ and let TR_F be the derived set of forward rules with target $NACs$. A NAC-consistent sequence*

$$\varnothing = G_{00} \overset{tr_S^*}{\Longrightarrow} G_{n0} \overset{g_n}{\hookrightarrow} G_0 \overset{tr_F^*}{\Longrightarrow} G_n$$

defined by pushout diagrams (1) *and* (3) *for $i = 1 \dots n$ with $G_0^C = \varnothing$, $G_0^T = \varnothing$ and inclusion $g_n : G_{n0} \hookrightarrow G_0$ is called* partially match consistent*, if diagram* (2) *commutes for all i, which means that the source component of the forward match $m_{i,F}$ is determined by the comatch $n_{i,S}$ of the corresponding step of the source sequence with $g_i = g_n \circ trace(G_{00} \overset{tr_S^*}{\Longrightarrow} G_{n0})_{i,n} = g_n \circ t_{n,S} \circ \dots \circ t_{i+1,S}$.*

$$\begin{array}{ccccccc} L_{i,S} & \overset{tr_{i,S}}{\hookrightarrow} & R_{i,S} & \hookrightarrow & L_{i,F} & \overset{tr_{i,F}}{\hookrightarrow} & R_{i,F} \\ {\scriptstyle m_{i,S}}\downarrow & (1) & \downarrow{\scriptstyle n_{i,S}} & (2) & {\scriptstyle m_{i,F}}\downarrow & (3) & \downarrow{\scriptstyle n_{i,F}} \\ G_{i-1,0} & \underset{t_{i,S}}{\hookrightarrow} & G_{i,0} & \underset{g_i}{\hookrightarrow} & G_0 & \hookrightarrow & G_{i-1} \quad \underset{t_{i,F}}{\hookrightarrow} \quad G_i \end{array}$$

Remark 5.1.17 (Partial Match Consistency)**.**

1. *If $g_n = id_{G_0}$, partial match consistency coincides with match consistency, i.e. $m_{i,F}^S = g_i^S \circ n_{i,S}^S$ for $g_i = trace(G_{00} \overset{tr_S^*}{\Longrightarrow} G_{n0})_{i,n} = t_{n,S} \circ \dots \circ t_{i+1,S}$.*
2. *For $n = 0$ the partially match consistent sequence is given by $g_0 : G_{00} \hookrightarrow G_0$.*

Definition 5.1.18 (Partial Source Consistency)**.** *A NAC-consistent forward sequence $G_0 \overset{tr_F^*}{\Longrightarrow} G_n$ is* partially source consistent*, if there is a source sequence $\varnothing = G_{00} \overset{tr_S^*}{\Longrightarrow} G_{n0}$ with inclusion $G_{n0} \overset{g_n}{\hookrightarrow} G_0$ such that $G_{00} \overset{tr_S^*}{\Longrightarrow} G_{n0} \overset{g_n}{\hookrightarrow} G_0 \overset{tr_F^*}{\Longrightarrow} G_n$ is partially match consistent.*

Analogous to source consistency (see Def. 5.1.12), the corresponding source sequence of a partially source consistent forward sequence is unique. Each match of the forward sequence fully determines the comatches of the source sequence and therefore, also the matches of the source sequence. This builds the basis for the on-the-fly construction of model transformations according to Thm. 5.1.22, where the source sequence is built up incrementally, and because of uniqueness we do not have to check further candidates.

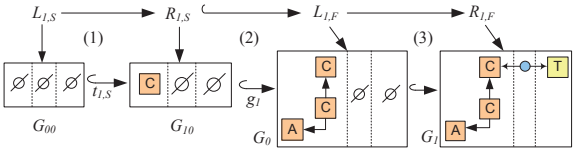


Figure 5.7: Step 1 of the partially match-consistent sequence

Example 5.1.19 (Partial Match and Source Consistency)**.** *Let us consider a sequence starting with triple graph G_0 (depicted in the centre of 5.7) which represents a class diagram consisting of a class with a subclass that has a primary attribute. The figures are simplified by omitting the attribute values.*

In the first step ($i = 1$), shown in Fig. 5.7, we apply rule $tr_{1,S} = Class2Table_S$ to the empty start graph G_{00} yielding the source graph G_{10} which contains one class. Obviously, G_{10} is included in G_0. Hence, diagram (2) commutes for step 1. The corresponding forward rule $tr_{1,F} = Class2Table_F$ is applied to G_0 and maps the class node to a table node, resulting in G_1. For step $i = 2$ (not depicted), we apply the same source, i.e. $tr_{2,S} = Class2Table_S$, to graph G_{10}. Thus, a separate class node is inserted and the resulting graph is G_{20} shown on the left of Fig. 5.8. Again, G_{20} is included in G_0, which is included in G_1, and diagram (2) for step 2 commutes. The corresponding forward rule $tr_{2,F} = Class2Table_F$ is applied to G_1, resulting in G_2, where the two class nodes are connected to two separate table nodes.

In the third step ($i = 3$), shown in Fig. 5.8, we apply the source rule $tr_{3,S} = PrimaryAttr2Column_S$, and add an attribute to G_{20}, resulting in source graph G_{30}. This graph is included in G_0, which in turn is included in G_2. Diagram (2) commutes for step 3. The application of the corresponding forward rule $tr_{3,F} = Attr2Column_F$ at the comatch of $tr_{3,S}$ yields G_3, where now the attribute is mapped to a column of the lower table.

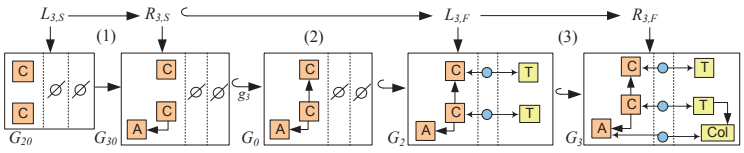


Figure 5.8: Step 1 of the partially match-consistent sequence

Since for each considered step, diagram (2) of Def. 5.1.16 commutes, we conclude that sequence $\emptyset = G_{00} \overset{tr_{1,S}}{\Longrightarrow} G_{10} \overset{tr_{2,S}}{\Longrightarrow} G_{20} \overset{tr_{3,S}}{\Longrightarrow} G_{30} \overset{g_n}{\hookrightarrow} G_0 \overset{tr_{1,F}}{\Longrightarrow} G_1 \overset{tr_{2,F}}{\Longrightarrow} G_2 \overset{tr_{3,F}}{\Longrightarrow} G_3$ is partially match consistent. Hence, the forward sequence $G_0 \overset{tr_{1,F}}{\Longrightarrow} G_1 \overset{tr_{2,F}}{\Longrightarrow} G_2 \overset{tr_{3,F}}{\Longrightarrow} G_3$

is partially source consistent. Note that the forward sequence, although being partially source consistent, cannot be extended to a complete source consistent sequence. There is no new match for some $tr_{4,F}$ leading to a partially source consistent sequence. The reason is that the subclass of the class diagram was translated by the rule $Class2Table_F$ instead of $Subclass2Table_F$, such that the edge of type "parent" was not matched and cannot be translated separately by any rule. In order to derive a complete source consistent sequence we have to backtrack leading to the source consistent sequence in Ex. 5.1.24.

In order to provide an improved construction of source consistent forward sequences we characterize valid matches by introducing the following notion of forward consistent matches. The formal condition of a forward consistent match is given by a pullback diagram where both matches satisfy the corresponding NACs. Intuitively, it specifies that the effective elements of the forward rule are matched for the first time in the forward sequence (see Interpretation 1 below).

Definition 5.1.20 (Forward Consistent Match)**.** *Given a partially match consistent sequence $\varnothing = G_{00} \overset{tr_S^*}{\Longrightarrow} G_{n-1,0} \overset{g_n}{\hookrightarrow} G_0 \overset{tr_F^*}{\Longrightarrow} G_{n-1}$ then a match $m_{n,F} : L_{n,F} \to G_{n-1}$ for $tr_{n,F} : L_{n,F} \to R_{n,F}$ is called* forward consistent *if there is a source match $m_{n,S}$ such that diagram (1) is a pullback and the matches $m_{n,F}$ and $m_{n,S}$ satisfy the corresponding target and source $NACs$, respectively.*

$$\begin{array}{ccccc} L_{n,S} & \hookrightarrow & R_{n,S} & \hookrightarrow & L_{n,F} \\ {\scriptstyle m_{n,S}}\downarrow & & (1) & & \downarrow{\scriptstyle m_{n,F}} \\ G_{n-1,0} & \underset{g_{n-1}}{\hookrightarrow} & G_0 & \hookrightarrow & G_{n-1} \end{array}$$

Interpretation 1. *The pullback property of (1) means that the intersection of the match $m_{n,F}(L_{n,F})$ and the source graph $G_{n-1,0}$ constructed so far is equal to $m_{n,F}(L_{n,S})$, the match restricted to $L_{n,S}$, i.e. we have*

$$(2) : m_{n,F}(L_{n,F}) \cap G_{n-1,0} = m_{n,F}(L_{n,S}).$$

This condition can be checked easily and $m_{n,S} : L_{n,S} \to G_{n-1,0}$ is uniquely defined by restriction of $m_{n,F} : L_{n,F} \to G_{n-1}$. Furthermore, as a direct consequence of (2) we have

$$(3) : m_{n,F}(L_{n,F} \setminus L_{n,S}) \cap G_{n-1,0} = \varnothing.$$

On the one hand, the source elements of $L_{n,F} \setminus L_{n,S}$ - called effective elements - are the elements to be transformed by the next step of the forward transformation sequence. On the other hand, $G_{n-1,0}$ contains all elements that were matched by the preceding forward steps, because matches of the forward sequence coincide on the source part with comatches of the source sequence. Hence, condition (3) means that the effective elements were not matched before, i.e. they do not belong to $G_{n-1,0}$.

Example 5.1.21 (Forward Consistent Match)**.** *In the partial match consistent sequence from Ex. 5.1.19, all forward rule matches are forward consistent. Consider for example the situation in step 3, shown in Fig. 5.9, where all mappings have been indicated explicitly by equal numbers. We can see that $L_{3,F} \cap G_{20} = L_{3,S}$, which implies that Diagram (1)*

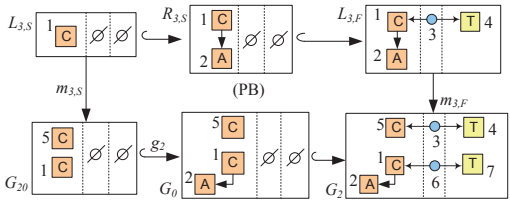


Figure 5.9: Forward consistent match in step 3

from Def. 5.1.20 is a pullback. Analogously, the matches from forward rules in steps 1 and 2 are also forward consistent.

The following on-the-fly construction, introduced in [EEHP09b, EEHP09a], incrementally extends partially source consistent sequences in order to derive complete source consistent sequences, which ensures, e.g., that all elements of the source model are translated exactly once. The source and forward sequences are synchronously built up. But still, the procedure may need to backtrack if the sequences cannot be extended to match consistent ones. Backtracking is also necessary in our case study as e.g. the rule "$Class2Table_F$" can be applied to class nodes that are subclasses as presented in Ex. 5.1.19. The procedure checks at each step whether a forward consistent match can be found and in the positive case extends the current source and forward sequences accordingly. Both, depth first search and breath first search can be performed. This allows us to filter the available matches such that matches that cannot lead to correct model transformations are rejected. Thus, the procedure checks partial source consistency on-the-fly and we do not have to analyse the resulting forward sequences for source consistency. In the case that all triple rules are creating on the source component termination is ensured for all possible source models. A constructed forward sequence is complete and source consistent if the derived triple graph $G_{n,0}$ of the source sequence is equal to the triple graph $(G^S \leftarrow \varnothing \rightarrow \varnothing)$ containing the source model G^S.

Theorem 5.1.22 (On-the-Fly Construction of Model Transformations)**.** *Given a triple graph G_0 with $G_0^C = G_0^T = \varnothing$, execute the following steps:*

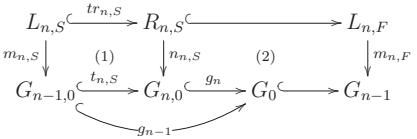
1. *Start with $G_{00} = \varnothing$ and $g_0 : G_{00} \hookrightarrow G_0$.*
2. *For $n > 0$ and an already computed partially source consistent sequence $s = seqG_0 \overset{tr_F^*}{\Longrightarrow} G_{n-1}$ with $\varnothing = G_{00} \overset{tr_S^*}{\Longrightarrow} G_{n-1,0}$ and embedding $g_{n-1} : G_{n-1,0} \hookrightarrow G_0$ find a (not yet considered) forward consistent match for some $tr_{n,F}$ leading to a partially source consistent sequence $G_0 \overset{tr_F^*}{\Longrightarrow} G_{n-1} \overset{tr_{n,F}}{\Longrightarrow} G_n$ with $G_{00} \overset{tr_S^*}{\Longrightarrow}$*

$G_{n-1,0} \overset{tr_{n,S}}{\Longrightarrow} G_{n0}$ and embedding $g_n : G_{n0} \hookrightarrow G_0$. If there is no such match, s cannot be extended to a source consistent sequence. Repeat until $g_n = id_{G_0}$ or no new forward consistent matches can be found.

3. *If the procedure terminates with $g_n = id_{G_0}$, then $G_0 \overset{tr_F^*}{\Longrightarrow} G_n$ is source consistent leading to a model transformation sequence $(G^S, G_0 \overset{tr_F^*}{\Longrightarrow} G_n, G^T)$ with G^S and G^T being the source and target models of G_0 and G_n.*

Proof. We have to show that this procedure is well-defined, i.e. that in Step 2, a forward consistent match leads to an extended partially source consistent sequence $G_0 \overset{tr_F^*}{\Longrightarrow} G_n$.

Given the situation as in Step 2 above, $(1) + (2)$ is a pullback because $m_{n,F}$ is forward consistent. The construction of pushout (1) leads to the source transformation sequence $G_{n-1,0} \overset{tr_{n,S}}{\Longrightarrow} G_{n0}$, embedding $g_n^S : G_{n,0}^S \hookrightarrow G_0^S$ with $g_n^S \circ t_{n,S}^S = g_{n-1}^S$ and $g_n^S \circ n_{n,S}^S = m_{n,F}^S$ due to (1) being a pushout and $(1) + (2)$ being a pullback over $\mathcal{M}$-morphisms. Here we use the fact that the category of attributed graphs $(\mathbf{AGraphs}, \mathcal{M})$ has effective unions according to Thm. 2 in [Her08b]. Finally, g_n^S defines the embedding $g_n : G_{n0} \hookrightarrow G_0$ with $g_n^C = \varnothing$ and $g_n^T = \varnothing$. Moreover, $G_{n-1,0} \overset{tr_{n,S}}{\Longrightarrow} G_{n0}$ and $G_{n-1} \overset{tr_{n,F}}{\Longrightarrow} G_n$ are NAC-consistent by assumption. Thus, $G_0 \overset{tr_F^*}{\Longrightarrow} G_n$ is partially source consistent.



□

Remark 5.1.23. *If the on-the-fly construction terminates with $g_n = id$ we have by Thm. 5.1.22 that the constructed forward transformation sequence is source consistent and specifies a model transformation sequence. Vice versa, if there is a source consistent forward transformation sequence for a given source model we can also ensure that the on-the-fly construction will compute one according to Thm. 5.2.2 in Sec. 5.2.1. This means that the on-the-fly construction is sound and complete in the sense that a source model is transformed into its corresponding target model via the on-the fly construction if and only if there is a model transformation sequence for this source model.*

The on-the-fly construction provides the basis for both, depth first search and breath first search. The choice is performed in step 2:

- *Depth First:* If we increase n after every iteration, and only decrease n by 1 if no more new forward consistent matches can be found, a depth-first search is performed.
- *Breadth First:* If we increase n only after all forward consistent matches for n are considered, the construction performs a breadth-first search.

Depending on the type of the model transformation, other search strategies may be reasonable. In Sec. 4 of [EEHP09b] we present how parallel independence of forward transformation steps can be analyed including the handling of partial source consistency in order to apply partial order reduction techniques that allow to improve the efficiency of the construction.

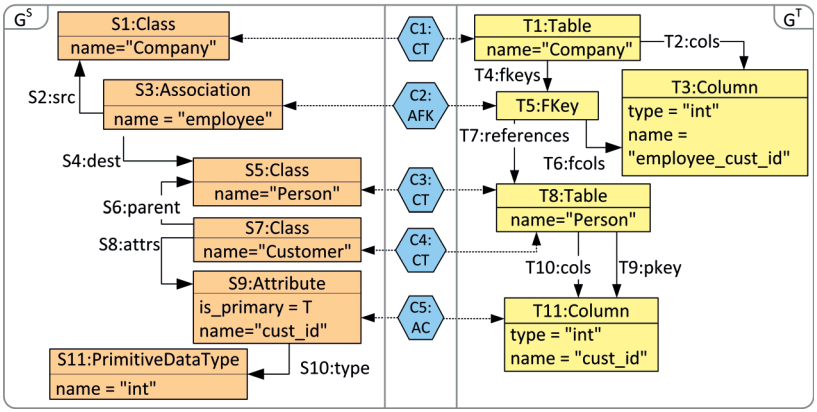


Figure 5.10: Triple graph G_5 of the example forward sequence

Table 5.1: Steps of Source Consistent Model Transformation

	Source Sequence Elements		Forward Sequence Elements	
Step	Matched	Created	Matched	Created
1		S1	S1	C1,T1
2		S5	S5	C3,T8
3	S5	S6,S7	S5-S7,C3,T8	C4
4	S7	S8-S11	S7-S11,C4,T8	C5,T9-T11
5	S1,S5	S2-S4	S1-S5,C1,C3,T1,T8	C2,T2-T7

Example 5.1.24 (Model Transformation Sequence based on Forward Rules)**.** *Based on the on-the-fly construction we derive the following source consistent forward sequence for our example class diagram shown in the source component of the triple graph in Fig. 5.10. An example for a source-consistent sequence is the model transformation* $(G^S = G_0^S, G_0 \overset{tr_F^*}{\Longrightarrow} G_5, G^T = G_5^T)$*, where* G_5 *(shown in Fig. 5.10) is generated by the forward sequence* $G_0 \overset{Class2Table_F}{\Longrightarrow} G_1 \overset{Class2Table_F}{\Longrightarrow} G_2 \overset{Subclass2Table_F}{\Longrightarrow} G_3 \overset{PrimaryAttr2Col_F}{\Longrightarrow} G_4 \overset{Association2FKey_F}{\Longrightarrow} G_5$*, and* G_0 *is generated by the corresponding source sequence*

$\varnothing \overset{tr_S^*}{\Longrightarrow} G_0$. *All elements in Fig. 5.10 are labelled with numbers. Table 5.1 specifies the matches and the created objects for each transformation step. Note that we cannot accidentally apply the rule* $Class2Table_F$ *at subclasses, because in this case the transformation will not become source consistent - the edge of the type "parent" will be missing.*

5.1.2 Model Transformation Based on Forward Translation Rules

As presented in Sec. 5.1.1 before the concept of model transformations based on source consistent forward sequences provides a formal basis for model transformations based on TGGs in general including formal results concerning the analysis of syntactical correctness, completeness and termination as they are presented in Sec. 5.2.1 following this section. But before, we present in this section how the formal condition *source consistency* can be equivalently encapsulated within forward rules by generating so-called *translation attributes*. This way, we can further extend the analysis capabilities in Sections. 5.2.2-5.2.2, improve the efficiency of the execution in Sec. 5.3) and finally, we can use standard and efficient graph transformation tools for these purposes.

The main result in this section shows that model transformations based on source consistent forward TGT-sequences are equivalent to those based on complete forward translation TGT-sequences as stated by Thm. 5.1.34. The control condition source consistency is ensured by the completeness of forward translation TGT-sequences, which are based on the generated forward translation rules. For this reason, the check of source consistency for forward TGT-sequences is reduced to a check whether the model is completely translated, i.e. all translation attributes are set to "T".

In many practical applications, model transformations are required to preserve the source model in order to use data base driven model repositories. For this reason, we presented in [HEGO10a] how the translation attributes can be externalized using the concept of triple graphs with interfaces. The translation attributes are equivalently replaced by external pointer structures such that the model transformation can be performed without any modification of the source model. This concept corresponds to the transformation algorithm in [SK08] which uses a separate set of translated elements and furthermore, it shows how the source steps within the on-the-fly construction in the previous section can be performed by an implementation by additional pointer structures as explained at the end of this section.

The extension of forward rules to forward translation rules is based on new attributes that control the translation process according to the source consistency condition. For each node, edge and attribute of a graph a new attribute is created and labelled with the prefix "tr". Given an attributed graph $AG = (G, D)$ and a family of subsets $M \subseteq G$ for nodes and edges, we call AG' a graph with translation attributes over AG if it extends AG with one Boolean-valued attribute $\mathtt{tr}\_x$ for each element x (node or edge) in M and one Boolean-valued attribute $\mathtt{tr}\_x\_a$ for each attribute associated to such an element x in M.

The family M together with all these additional translation attributes is denoted by Att_M. Note that we use the attribution concept of E-Graphs as presented in [EEPT06], where attributes are possible for nodes and edges.

Definition 5.1.25 (Family with Translation Attributes)**.** *Given an attributed graph $AG = (G, D)$ we denote by $|G| = (V_G^G, V_G^D, E_G^G, E_G^{NA}, E_G^{EA})$ the underlying family of sets containing all nodes and edges. Let $M \subseteq |G|$ with $(V_M^G, V_M^D, E_M^G, E_M^{NA}, E_M^{EA})$, then a* family with translation attributes *for (G, M) extends M by additional translation attributes and is given by $Att_M = (V_M^G, V_M^D, E_M^G, E^{NA}, E^{EA})$ with:*

- $E^{NA} = E_M^{NA} \cup \{\texttt{tr\_}x \mid x \in V_M^G\} \cup \{\texttt{tr\_}x\texttt{\_}a \mid a \in E_M^{NA}, src_G^{NA}(a) = x \in V_G^G\}$,
- $E^{EA} = E_M^{EA} \cup \{\texttt{tr\_}x \mid x \in E_M^G\} \cup \{\texttt{tr\_}x\texttt{\_}a \mid a \in E_M^{EA}, src_G^{EA}(a) = x \in E_G^G\}$.

Definition 5.1.26 (Graph with Translation Attributes)**.** *Given an attributed graph $AG = (G, D)$ and a family of subsets $M \subseteq |G|$ with $\{\mathbf{T}, \mathbf{F}\} \subseteq V_M^D$ and let Att_M be a family with translation attributes for (G, M). Then, $AG' = (G', D)$ is a graph with translation attributes over AG, where $|G'|$ is the gluing of $|G|$ and Att_M over M, i.e. the sets of nodes and edges are given by componentwise pushouts and the source and target functions of AG' are defined as follows:*

- $src_{G'}^G = src_G^G$, $trg_{G'}^G = trg_G^G$,
- $src_{G'}^X(z) = \begin{cases} src_G^X(z) & z \in E_G^X \\ x & z = \texttt{tr\_}x \text{ or } z = \texttt{tr\_}x\texttt{\_}a \end{cases}$ *for* $X \in \{NA, EA\}$,
- $trg_{G'}^X(z) = \begin{cases} trg_G^X(z) & z \in E_G^X \\ \mathbf{T} \text{ or } \mathbf{F} & z = \texttt{tr\_}x \text{ or } z = \texttt{tr\_}x\texttt{\_}a \end{cases}$ *for* $X \in \{NA, EA\}$.

$$\begin{array}{ccc} M & \hookrightarrow & Att_M \\ \downarrow & (PO) & \downarrow \\ |G| & \longrightarrow & |G'| \end{array}$$

Att_M^v, where $v = \mathbf{T}$ or $v = \mathbf{F}$, denotes a family with translation attributes where all attributes are set to v. Moreover, we denote by $AG \oplus Att_M$ that AG is extended by the translation attributes in Att_M i.e. $AG \oplus Att_M = (G', D)$ for $AG' = (G', D)$ defined above. Analogously, we use the notion $AG \oplus Att_M^v$ for translation attributes with value v and we use the short notation $Att^v(AG) := AG \oplus Att_{|G|}^v$.

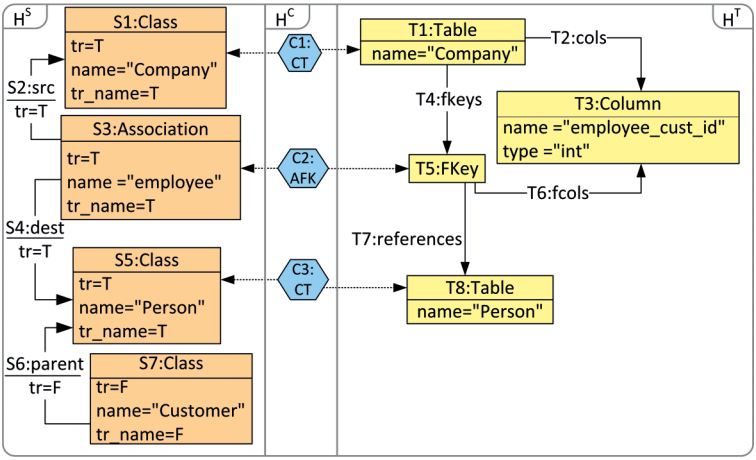


Figure 5.11: Triple graph with translation attributes

Example 5.1.27 (Triple Graph with Translation Attributes)**.** *Fig. 5.11 shows the triple graph $H = (H^S \leftarrow H^C \rightarrow H^T)$ which specifies a subgraph of the triple graph G in Fig. 5.10 that is extended by some translation attributes in the source component. The translation attributes with value "***T***" indicate that the owning elements have been translated during a model transformation sequence using forward translation rules, which are defined in Def. 5.1.28 hereafter. The remaining elements (edge S6, node S7 and the attribute "name" of S7) in the source component are still marked with translation attributes set to "***F***". These elements can still be matched and will become translated at later steps. The translation attributes are used to explicitly specify the elements which have been translated up to a specific step during the execution of a model transformation.*

The concept of forward translation rules, which we introduced in [HEOG10], extends the construction of forward rules by additional translation attributes in the source component. As described in Ex. 5.1.27, the translation attributes are used to keep track of the elements that have been translated so far. This way, we can ensure that each element in the source graph is not translated twice, but exactly once. At the beginning, the source model of a model transformation sequence is extended by translation attributes that are all set to "**F**" and they are set to "**T**" when their containing elements are translated by a forward translation rule. Whenever the model transformation stops at a model whose translation attributes are all set to "**T**", the sequence specifies a source consistent forward sequence by removing all translation attributes and a valid target model is obtained from the resulting triple graph. Due to the modification of the translation attributes, the rules are deleting and

thus, the triple transformations are extended from a single (total) pushout to the classical double pushout (DPO) approach [EEPT06]. We call these rules forward translation rules, because pure forward rules need to be controlled by additional control conditions, such as the source consistency condition in Sec. 5.1.1 for the on-the-fly construction.

Definition 5.1.28 (Forward Translation Rules with NACs)**.** *Given a triple rule* $tr = (L \to R)$*, the* forward translation rule *of* tr *is given by* $tr_{FT} = (L_{FT} \xleftarrow{l_{FT}} K_{FT} \xrightarrow{r_{FT}} R_{FT})$ *defined as follows using the forward rule* $(L_F \xrightarrow{tr_F} R_F)$ *and the source rule* $(L_S \xrightarrow{tr_S} R_S)$ *of* tr*, where we assume w.l.o.g. that* tr *is an inclusion:*

- $L_{FT} = L_F \oplus Att^{\mathbf{T}}_{L_S} \oplus Att^{\mathbf{F}}_{R_S \setminus L_S}$
- $K_{FT} = L_F \oplus Att^{\mathbf{T}}_{L_S}$
- $R_{FT} = R_F \oplus Att^{\mathbf{T}}_{L_S} \oplus Att^{\mathbf{T}}_{R_S \setminus L_S}$
  $= R_F \oplus Att^{\mathbf{T}}_{R_S},$
- l_{FT} *and* r_{FT} *are the induced inclusions.*

Moreover, for each NAC $n : L \to N$ *of* tr *we define a forward translation NAC* $n_{FT} : L_{FT} \to N_{FT}$ *of* tr_{FT} *as inclusion with* $N_{FT} = (L_{FT} +_L N) \oplus Att^{\mathbf{T}}_{N_S \setminus L_S}$.

Remark 5.1.29. *Note that* $(L_{FT} +_L N)$ *is the union of* L_{FT} *and* N *with shared* L *and for a target NAC* n *the forward translation NAC* n_{FT} *does not contain any translation attributes because* $N_S = L_S$.

Example 5.1.30 (Derived Forward Translation Rules)**.** *Figure 5.12 shows the derived forward translation rule "*$Subclass2Table_{FT}$*" for the triple rule "*$Subclass2Table$*" in Fig. 5.4. Note that we abbreviate "*$\texttt{tr}\_x$*" for an item (node or edge)* x *by "*$\texttt{tr}$*" and "*$\texttt{tr}\_x\_a$*" by "*$\texttt{tr}\_type(a)$*" in the figures to increase readability. The compact notation of forward translation rules specifies the modification of translation attributes by "*$[\mathbf{F} \Rightarrow \mathbf{T}]$*", meaning that the attribute is matched with the value "*$\mathbf{F}$*" and set to "*$\mathbf{T}$*" during the transformation step. The detailed complete notation of a forward translation rule is shown on the right of Fig. 5.12 for "*$Subclass2Table_{FT}$*".*

Fig. 5.13 shows the forward translation rule with NACs "$PrimaryAttr2Column_{FT}$*" derived from the triple "*$PrimaryAttr2Column$*", which is a* TGG*-triple rule of our case study CD2RDBM (cf. Fig. 5.5). According to Def. 5.1.28 the source elements of the triple rule are extended by translation attributes and changed by the rule from "*$\mathbf{F}$*" to "*$\mathbf{T}$*", if the owning elements are created by the triple rule. Furthermore, the forward translation rule contains both, the source and the target NACs of the* TGG*-triple rule, where the NAC-only elements in the source NACs are extended by translation attributes set to "*$\mathbf{T}$*". Thus, the source NACs concern only elements that have been translated so far.*

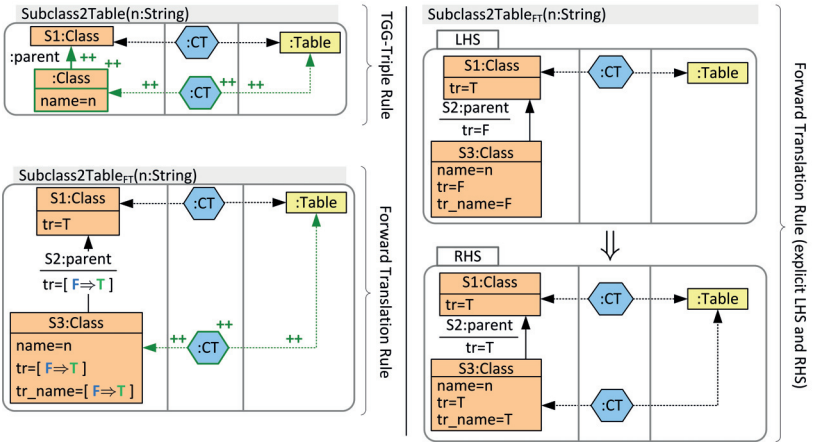


Figure 5.12: Forward translation rule $Subclass2Table_{FT}(n : String)$

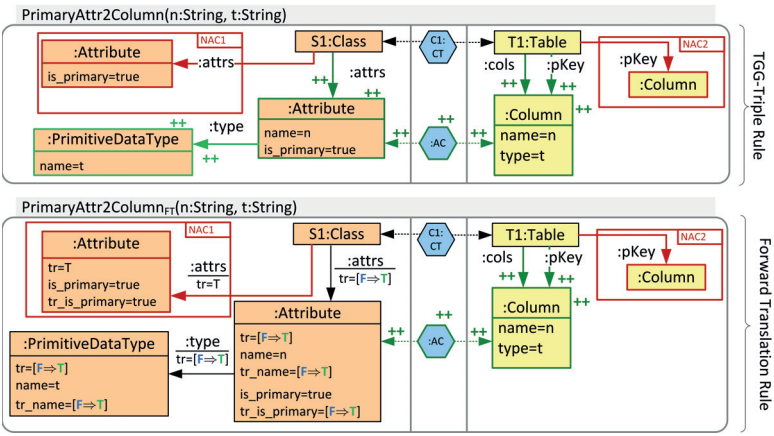


Figure 5.13: Forward translation rule with NACs

From the application point of view a model transformation should be injective on the structural part, i.e. the transformation rules are applied along matches that do not identify structural elements. But it would be too restrictive to require injectivity of the matches also on the data and variable nodes, because we must allow that two different variables are mapped to the same data value. For this reason we introduce the notion of almost

injective matches, which requires that matches are injective except for the data value nodes. This way, attribute values can still be specified as terms within a rule and matched non-injectively to the same value.

Definition 5.1.31 (Almost Injective Match and Completeness)**.** *An attributed triple graph morphism $m : L \to G$ is called* almost injective*, if it is non-injective at most for the set of variables and data values in L_{FT}. A forward translation sequence $G_0 \overset{tr^*_{FT}}{\Longrightarrow} G_n$ with almost injective matches is called* complete *if no further forward translation rule is applicable and G_n is completely translated, i.e. all translation attributes of G_n are set to true ("$\mathbf{T}$").*

Now, we define model transformations based on forward translation rules in the same way as for forward rules in Def. 5.1.15, where source consistency of the forward sequence is replaced by completeness of the forward translation sequence.

Definition 5.1.32 (Model Transformation Based on Forward Translation Rules)**.** *A* model transformation sequence $(G^S,\ G'_0 \overset{tr^*_{FT}}{\Longrightarrow} G'_n, G^T)$ *based on forward translation rules consists of a source graph G^S, a target graph G^T, and a complete* TGT*-sequence $G'_0 \overset{tr^*_{FT}}{\Longrightarrow} G'_n$ with almost injective matches, $G'_0 = (Att^{\mathbf{F}}(G^S) \leftarrow \varnothing \rightarrow \varnothing)$ and $G'_n = (Att^{\mathbf{T}}(G^S) \leftarrow G^C \rightarrow G^T)$.*
A model transformation $MT : VL_{S0} \Rrightarrow VL_{T0}$ *based on forward translation rules with NACs is defined by all model transformation sequences as above with $G^S \in VL_{S0}$ and $G^T \in VL_{T0}$. All the corresponding pairs (G^S, G^T) define the* model transformation relation $MTR_{FT} \subseteq VL_{S0} \times VL_{T0}$ *based on forward translation rules. The model transformation is* terminating *if there are no infinite* TGT*-sequences via forward translation rules and almost injective matches starting with $G'_0 = (Att^{\mathbf{F}}(G^S) \leftarrow \varnothing \rightarrow \varnothing)$ for some source graph G^S.*

Example 5.1.33 (Model Transformation via Forward Translation Rules)**.** *Fig. 5.14 shows the resulting triple graph of a forward translation sequence starting with the same source model as in Ex. 5.1.24 in Sec. 5.1.1. This time the execution starts by extending the source model G^S with translation attributes according to Def. 5.1.32, i.e. $G'_0 = (Att^{\mathbf{F}}(G^S) \leftarrow \varnothing \rightarrow \varnothing)$. We can execute the forward translation sequence using the same sequence of rules as in Ex. 5.1.24 but instead of forward rules we apply their corresponding forward translation rules. Thus the sequence is $G'_0 \overset{Class2Table_{FT}}{\Longrightarrow} G'_1 \overset{Class2Table_{FT}}{\Longrightarrow} G'_2 \overset{Subclass2Table_{FT}}{\Longrightarrow} G'_3 \overset{PrimaryAttr2Col_{FT}}{\Longrightarrow} G'_4 \overset{Association2FKey_{FT}}{\Longrightarrow} G'_5$, with G'_5 being the graph G in Fig. 5.14. Now, the triple graph G'_5 is completely translated, because all translation attributes are set to "$\mathbf{T}$". No further forward translation rule is applicable and we derive the resulting target model G^T by restricting G'_5 to its target component, i.e. $G^T = G'^T_5$. According to the equivalence of the model transformation*

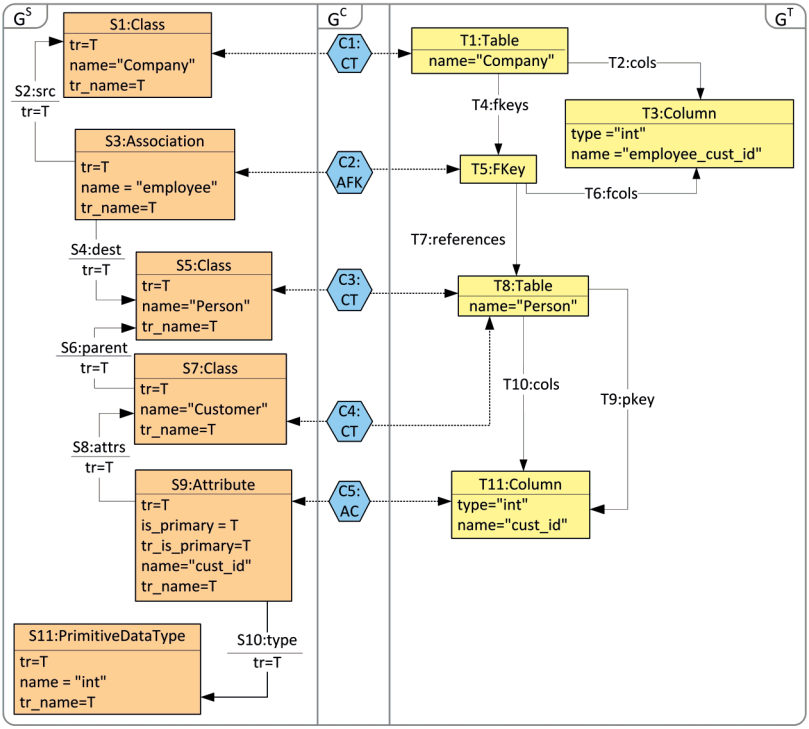


Figure 5.14: Triple graph instance with translation attributes for *CD2RDBM*

concepts based on forward and forward translation rules in Thm. 5.1.34 below we can further conclude that G^T is also a valid result using the on-the-fly construction for model transformations based on forward rules in Sec. 5.1.1.

By Thm. 5.1.34 below we show that the model transformation sequences based on forward translation rules are one-to-one with model transformation sequences based on forward rules, i.e. based on source consistent forward sequences. For this reason, we can equivalently use both concepts and chose one of them depending on the particular needs. While the concept based on source consistency shows advantages in formal proofs the concept based on forward translation rules shows advantages concerning analysis and efficiency as we will show in Sec. 5.2 and 5.3.

Theorem 5.1.34 (Complete Forward Translation Sequences with NACs)**.** *Given a source model $G^S \in VL_{S0}$, then the following are equivalent for almost injective matches.*

1. $\exists$ *a model transformation sequence* $(G^S, G_0 \overset{tr_F^*}{\Longrightarrow} G_n, G^T)$ *based on forward rules with* $G_n = (G^S \leftarrow G^C \rightarrow G^T)$

2. $\exists$ *a model transformation sequence* $(G^S, G'_0 \overset{tr_{FT}^*}{\Longrightarrow} G'_n, G^T)$ *based on forward translation rules with* $G'_n = (Att^{\mathbf{T}}(G^S) \leftarrow G^C \rightarrow G^T)$.

Moreover, the model transformation relation MTR_F *for the model transformation based on forward rules coincides with the model transformation relation* MTR_{FT} *for the model transformation based on forward translation rules, i.e.* $MTR_F = MTR_{FT}$.

In order to proof Thm. 5.1.34 above we use Lem. 5.1.35 below concerning the equivalence of single transformation steps. The proof of Lem. 5.1.35 is given by the proof of Fact 1 in [HEGO10c].

Lemma 5.1.35 (Forward translation step)**.** *Let* TR *be a set of triple rules with* $tr_i \in TR$ *and let* TR_F *be the derived set of forward rules. Given a partially match consistent forward sequence* $\varnothing = G_{00} \overset{tr_S^*}{\Longrightarrow} G_{i-1,0} \overset{g_{i-1}}{\hookrightarrow} G_0 \overset{tr_F^*}{\Longrightarrow} G_{i-1}$ *and a corresponding forward translation sequence* $G'_0 \overset{tr_{FT}^*}{\Longrightarrow} G'_{i-1}$*, both with almost injective matches, such that* $G'_{i-1} = G_{i-1} \oplus Att^{\mathbf{F}}_{G_0 \setminus G_{i-1,0}} \oplus Att^{\mathbf{T}}_{G_{i-1,0}}$*. Then the following are equivalent:*

1. $\exists$ TGT-*step* $G_{i-1} \overset{tr_{i,F}, m_{i,F}}{\Longrightarrow} G_i$ *with forward consistent match* $m_{i,F}$

2. $\exists$ *translation* TGT-*step* $G'_{i-1} \overset{tr_{i,FT}, m_{i,FT}}{\Longrightarrow} G'_i$

and we have $G'_i = G_i \oplus Att^{\mathbf{F}}_{G_0 \setminus G_{i,0}} \oplus Att^{\mathbf{T}}_{G_{i,0}}$.

Proof of Thm. 5.1.34. We first show the equivalence of the sequences disregarding the NACs.

1. $\Leftrightarrow G_0 \overset{tr_{1,F}, m_{1,F}}{\Longrightarrow} G_1 \overset{tr_{2,F}, m_{2,F}}{\Longrightarrow} G_2 \ldots \overset{tr_{n,F}, m_{n,F}}{\Longrightarrow} G_n$ with $G_n^S = G^S$, where each match is forward consistent according to Thm. 5.1.22.
2. $\Leftrightarrow G'_0 \overset{tr_{1,FT}, m_{1,FT}}{\Longrightarrow} G'_1 \overset{tr_{2,FT}, m_{2,FT}}{\Longrightarrow} G'_2 \ldots \overset{tr_{n,FT}, m_{n,FT}}{\Longrightarrow} G'_n$ is complete.

Disregarding the NACs, it remains to show that $G_0'^S = Att^{\mathbf{F}}(G^S)$ and $G_n'^S = Att^{\mathbf{T}}(G^S)$.
We apply Lemma 5.1.35 for $i = 0$ with $G_{0,0} = \varnothing$ up to $i = n$ with $G_{n,0} = G_0$ and using $G_0^S = G^S$ we derive:
$G_0'^S = G_0^S \oplus Att^{\mathbf{T}}_{G_{0,0}} \oplus Att^{\mathbf{F}}_{G_0^S \setminus G_{0,0}^S} = G_0^S \oplus Att^{\mathbf{F}}_{G_0^S} = G^S \oplus Att^{\mathbf{F}}_{G^S} = Att^{\mathbf{F}}(G_S)$.
$G_n'^S = G_n^S \oplus Att^{\mathbf{T}}_{G_{n,0}^S} \oplus Att^{\mathbf{F}}_{G_0^S \setminus G_{n,0}^S} = G_n^S \oplus Att^{\mathbf{T}}_{G_{n,0}^S} = G^S \oplus Att^{\mathbf{T}}_{G^S} = Att^{\mathbf{T}}(G^S)$.

Now, we show that the single steps are also NAC consistent.

For each step, we have transformations $G_{i-1,0} \xRightarrow{tr_{i,S},m_{i,S}} G_{i,0}$, $G_{i-1} \xRightarrow{tr_{i,F},m_{i,F}} G_i$, $G'_{i-1} \xRightarrow{tr_{i,FT},m_{i,FT}} G'_i$ with $G'_{i-1} = G_{i-1} \oplus Att^{\mathbf{F}}_{G_0 \setminus G_{i-1,0}} \oplus Att^{\mathbf{T}}_{G_{i-1,0}}$, $G'_i = G_i \oplus Att^{\mathbf{F}}_{G_0 \setminus G_{i,0}} \oplus Att^{\mathbf{T}}_{G_{i,0}}$, and $m_{i,FT}|_{L_{i,F}} = m_{i,F}$.

For a target NAC $n : L_i \to N$, we have to show that $m_{i,F} \models n$ iff $m_{i,FT} \models n_{FT}$, the corresponding forward translation NAC. If $m_{i,FT} \not\models n_{FT}$, we find a monomorphism q' with $q' \circ n_{FT} = m_{i,FT}$. Since $n = n_{FT}|_N$, define $q = q'|_N$ and it follows that $q \circ n = m_{i,F}$, i.e. $m_{i,F} \not\models n$. Vice versa, if $m_{i,F} \not\models n$, we find a monomorphism q with $q \circ n = m_{i,F}$. Since $N_S = L_{i,S}$, we do not have any additional translation attributes in N_{FT}. Thus $m_{i,FT}$ can be extended by q to $q' : N_{FT} \to G'_{i-1}$ such that $m_{i,FT} \not\models n_{FT}$.

Similarly, we have to show that for a source NAC $n : L \to N$, $m_{i,S} \models n$ iff $m_{i,FT} \models n_{FT}$. As for target NACs, if $m_{i,FT} \not\models n_{FT}$, we find a monomorphism q' with $q' \circ n_{FT} = m_{i,FT}$ and for the restriction to $L_{i,S}$ and N it follows that $q \circ n = m_{i,S}$, i.e. $m_{i,S} \not\models n$. Vice versa, if $m_{i,S} \not\models n$, we find a monomorphism q with $q \circ n = m_{i,S}$. Now define q' with $q'(x) = m_{i,FT}(x)$ for $x \in L_{FT}$, $q'(x) = q(x)$ for $x \in N \setminus L_i$, and for each $x \in N_S \setminus L_{i,S}$ we have that $q(x) \in G_{i-1,0}$. From the above characterization of G'_{i-1} it follows that the corresponding translation attributes `tr_`x and `tr_`x_a are set to $\mathbf{T}$ in G'_{i-1}. Thus, q' is well-defined and $q' \circ n_{FT} = m_{i,FT}$, i.e. $m_{i,FT} \not\models n_{FT}$.

The equality of the model transformation relations follows by the equality of the pairs (G^S, G^T) in the model transformation sequences in both cases. □

Remark 5.1.36. *It can be shown that the model transformation relation MTR defined by the triple rules TR coincides with the relations MTR_F and MTR_{FT} of the model transformations based on forward and forward translation rules TR_F and TR_{FT}, respectively.*

Applying a rule according to the DPO approach involves the check of the gluing condition in general. However, in the case of forward translation rules and almost injective matches we have that the gluing condition is always satisfied. This means that the condition does not have to be checked during the execution of the model transformation.

Fact 5.1.37 (Gluing Condition for Forward Translation Rules)**.** *Let tr_{FT} be a forward translation rule and $m_{FT} : L_{FT} \to G$ be an almost injective match, then the gluing condition is satisfied, i.e. there is the transformation step $G \xRightarrow{tr_{FT},m_{FT}} H$.*

Proof. According to Def. 9.8 in [EEPT06] we need to check that $DP \cup IP \subseteq GP$. First of all, the set IP may only contain data elements by the restriction of the match, which are in GP. Furthermore, the set DP does only contain nodes. The rule is only deleting on attribution edges and thus, $DP \cup IP \subseteq GP$. □

During the execution of a model transformation the given source model may be simultaneously used by other applications within an MDA environment and therefore, the model transformation should not modify the source model. Considering our case study, the model

transformation transforms class diagrams to data base tables. However, the class diagram may be additionally used for documenting the system structure and thus, should be available unchanged for the software development groups. Furthermore, other interrelated models may rely on a synchronized connection to the class diagram, e.g. a synchronization with corresponding block diagrams is common in the automotive domain as presented in [GW09].

For this reason, we presented in Sec. 5 of [HEGO10a] how the concept of model transformations based on forward translation rules with translation attributes can be equivalently implemented using a marking structure that points to the handled elements of the source model leaving the source model itself unchanged. This way, the additional structure necessary for ensuring the syntactical correctness and completeness of the model transformation is externalized from the source model and kept separately. More precisely, a triple graph consisting of the source, correspondence and target model is extended by an additional triple graph, called interface graph, which specifies the elements of the source model that have been translated so far. This means that the Boolean valued translation attributes are represented by the presence and absence of elements in the interface graph. Moreover, as shown by the equivalence result we also have that model transformation sequences based on forward translation rules with interfaces are equivalent to those constructed by the on-the-fly construction. In fact, both approaches separately construct the source and forward steps. However, the on-the-fly construction shows advantages in the handling of NACs, as they are distributed separately to the source and forward steps, while they have to be kept together for the transformation steps with interfaces. This implies that NACs have to be specified as pairs of triple graphs while the left and right hand sides of a rule form triple graphs with interfaces making this concept more complex.

All together, the concept based on interfaces presents one possibility how the source steps within the on-the-fly construction can be handled by an implementation using an additional pointer structure. The construction of model transformations itself remains unchanged compared to the on-the-fly construction in the previous section.

5.2 Analysis

Model transformations based on TGGs as presented in Sec. 5.1 before provide an excellent framework for analysing and verifying a major part of the properties that may have to be ensured in an application scenario regarding the first dimension of challenges for model transformations - the functional dimension - presented at the beginning of this chapter. This section presents powerful analysis techniques that are based on the introduced model transformation concepts while Sec. 5.3 thereafter focuses mainly on the second dimension of challenges - the non-functional dimension.

As first main results we show in Sec. 5.2.1 that the presented approaches for model transformations ensure syntactical correctness and completeness (see Thms. 5.2.1 and 5.2.2 and Cor. 5.2.6). Moreover, we provide a sufficient condition for termination (see Thm. 5.2.4 and Cor. 5.2.6), which is often satisfied for practical applications resp. can be usually achieved with minor efforts.

In Sec. 5.2.2 we furthermore show as a second group of main results how functional behaviour of model transformations can be efficiently analysed (see Thms. 5.2.27 and 5.2.31) with automated tool support. Moreover, we present how model transformations based on TGGs are analysed with respect to information preservation (see Thm. 5.2.34 and 5.2.37) based on the developed techniques before. Information preservation is one aspect relevant for the bidirectional characteristics of model transformations and thus, already concerns the non-functional dimension of challenges

5.2.1 Correctness, Completeness and Termination

As central challenges for model transformations they have to ensure syntactical correctness and completeness including termination. As a main advantage compared to other approaches for model transformation we can generally ensure syntactical correctness and completeness for the presented approaches in Sec. 5.1 (see Thms. 5.2.1 and 5.2.2 and Cor. 5.2.6) and require a relatively weak condition for ensuring termination (see Thm. 5.2.4 and Cor. 5.2.6), which is satisfied in many cases.

Syntactical Correctness of a model transformation based on TGGs states that each successful execution of a model transformation starting with a valid source model G^S yields a target model G^T which exactly corresponds to G^S according to the language of integrated models VL generated by the given TGG. Completeness means that all valid source models can be transformed. Moreover, we do not only show that our model transformations are left total with respect to source models, but they are also right total. This means that for each valid target model G^T there is a source model, which can be transformed into G^T. By Thm. 5.2.1 below we first show these results for model transformations based on forward rules according to Sec. 5.1.1 and thereafter also show that the on-the-fly construction as well as the concept of model transformation based on forward translation rules ensure these properties.

Theorem 5.2.1 (Syntactical Correctness and Completeness)**.** *Each model transformation $MT : VL_{S0} \Rrightarrow VL_{T0}$ based on forward rules is*

- syntactically correct*, i.e. for each model transformation sequence $(G^S, G_0 \overset{tr^*_F}{\Longrightarrow} G_n, G^T)$ there is $G \in VL$ with $G = (G^S \leftarrow G^C \rightarrow G^T)$ implying further that $G^S \in VL_S$ and $G^T \in VL_T$, and it is*

- complete, *i.e. for each* $G^S \in VL_S$ *there is* $G = (G^S \leftarrow G^C \rightarrow G^T) \in VL$ *with a model transformation sequence* $(G^S, G_0 \overset{tr_F^*}{\Longrightarrow} G_n, G^T)$. *Vice versa, for each* $G^T \in VL_T$ *there is* $G = (G^S \leftarrow G^C \rightarrow G^T) \in VL$ *with a model transformation sequence* $(G^S, G_0 \overset{tr_F^*}{\Longrightarrow} G_n, G^T)$.

Proof. The syntactical correctness and completeness results are based on the proof of Thm. 3 in [EEH08c]. Note that Thm. 3 in [EEH08c] states a weaker result of correctness and completeness for source consistent forward transformations. However, the proof is based on the composition and decomposition result for triple graph transformation sequences given by Thm. 1 in [EEE+07] and in the general form including NACs by Thm. 5.1.14.

Now, given a model transformation sequence based on forward translation rules $(G^S, G_0' \overset{tr_{FT}^*}{\Longrightarrow} G_n', G^T)$ we have by Thm. 5.1.34 that there is model transformation sequence based on forward rules $(G^S, G_0 \overset{tr_F^*}{\Longrightarrow} G_n, G^T)$, which means by definition that $G_0 \overset{tr_F^*}{\Longrightarrow} G_n$ is source consistent. Source consistency implies that there is a source sequence $\varnothing \overset{tr_S^*}{\Longrightarrow} G_{n,0} = G_0$ such that $\varnothing \overset{tr_S^*}{\Longrightarrow} G_{n,0} \overset{tr_F^*}{\Longrightarrow} G_n$ is match consistent and can be composed to the triple sequence $\varnothing \overset{tr^*}{\Longrightarrow} G_n \in VL$ by the composition result in Thm. 5.1.14. This means that the model transformation based on forward rules is syntactically correct.

Vice versa, given $G^S \in VL_S$ (resp. $G^T \in VL_T$) we have that there is a $G_n \in VL$ with $G^S = G_n^S$ (resp. $G^T = G_n^T$) and a TGT-transformation sequence $\varnothing \overset{tr^*}{\Longrightarrow} G_n$. Using the decomposition result in Thm. 5.1.14 we derive the model transformation sequence based on forward rules $(G^S, G_0 \overset{tr_F^*}{\Longrightarrow} G_n, G^T)$, i.e. the model transformation is complete.

□

The on-the-fly construction is syntactically correct, which means that if it terminates both the source and target models of the resulting model transformation sequence correspond to each other according to the language of integrated models VL generated by the TGG-triple rules. Moreover, it is also complete, which means that for any source model of the language VL_S the procedure can find a model transformation sequence and vice versa, for any target model G^T there is a source model for which the procedure can find a model transformation sequence resulting in G^T.

Theorem 5.2.2 (Syntactical Correctness and Completeness of the On-The-Fly construction)**.** *The on-the-fly construction is syntactically correct and complete, i.e.:*

- Syntactical Correctness: *If the on-the-fly construction terminates with* $g_n = id_{G_0}$, *then the resulting model transformation* $(G^S, G_0 \overset{tr_F^*}{\Longrightarrow} G_n, G^T)$ *is syntactically correct, i.e. there is* $G \in VL$ *with* $G = (G^S \leftarrow G^C \rightarrow G^T)$ *implying further that* $G^S \in VL_S$ *and* $G^T \in VL_T$.

- Completeness: *For each $G^S \in VL_S$ there exists $G^T \in VL_T$ with a model transformation $(G^S, G_0 \overset{tr_F^*}{\Longrightarrow} G_n, G^T)$, which can be obtained by the on-the-fly construction. Vice versa, for each $G^T \in VL_T$ there exists $G^S \in VL_S$ with a model transformation sequence $(G^S, G_0 \overset{tr_F^*}{\Longrightarrow} G_n, G^T)$, which can be obtained by the on-the-fly construction.*

Proof. **Syntactical Correctness:** If the procedure terminates with a source consistent forward transformation $G_0 \overset{tr_F^*}{\Longrightarrow} G_n$ and a corresponding source transformation $\varnothing = G_{00} \overset{tr_S^*}{\Longrightarrow} G_{n0} = G_0$ then there is a TGT-sequence $\varnothing = G_{00} \overset{tr^*}{\Longrightarrow} G_n$ with $G_0^S = G_n^S = G^S$ and $G_n^T = G^T$ and by Def. 5.1.7 $G^S \in VL_S$ and $G^T \in VL_T$.

Completeness: $G^S \in VL_S$ implies that there is a TGT-transformation sequence $\varnothing = G_{00} \overset{tr^*}{\Longrightarrow} G_n$ with $G_n^S = G^S$ and $tr^* = (tr_i)_{i=1\ldots n}$, which can be decomposed by Thm. 5.1.14 into a match-consistent sequence $G_{00} \overset{tr_S^*}{\Longrightarrow} G_{n0} = G_0 \overset{tr_F^*}{\Longrightarrow} G_n$ with matches $m_{i,S}$ and $m_{i,F}$. Vice versa, given $G^T \in VL_T$ we analogously have that there is a TGT-transformation sequence $\varnothing = G_{00} \overset{tr^*}{\Longrightarrow} G_n$ with $G^S = G_n^S$ and $G^T = G_n^T$ that can be decomposed into a match consistent sequence $G_{00} \overset{tr_S^*}{\Longrightarrow} G_{n0} = G_0 \overset{tr_F^*}{\Longrightarrow} G_n$.

In both cases, the on-the-fly construction starts with $\varnothing = G_{00}$ and $g_0 : G_{00} \hookrightarrow G_0$. In Step 2, for $i = 1, \ldots n$ we have a partially match consistent sequence $\varnothing = G_{00} \overset{tr_S^{1\ldots i}}{\Longrightarrow} G_{i0} \overset{g_i}{\hookrightarrow} G_0 \overset{tr_F^{1\ldots i}}{\Longrightarrow} G_i$. Choose $tr_{i+1,F}$ as the next rule in the forward sequence with match $m_{i+1,F}$. For the source match $m_{i+1,S}$, (1) is a pushout and since the original sequence is source consistent $m_{i+1,F}$ is uniquely determined by $n_{i+1,S}$, which means that there is an inclusion $g_{i+1} : G_{i+1,0} \hookrightarrow G_i$ such that $m_{i+1,F}^S = g_{i+1}^S \circ n_{i+1}^S$ and $g_i^S = g_{i+1}^S \circ t_{i+1,S}^S$. With (1) being both pushout and pullback, and g_{i+1} and $G_0 \hookrightarrow G_i$ being monomorphisms we have that $(1) + (2)$ is a pullback, leading to the fact that $m_{i+1,F}$ is forward consistent. After performing all n steps we have $g_n = id_{G_0}$ such that the termination criterion of the the on-the-fly construction is satisfied and the procedure stops leading to the target model $G^T = G_n^T$ for the source model $G^S = G_0^S$.

$$\begin{array}{ccccc} L_{i+1,S} & \overset{tr_{i+1,S}}{\hookrightarrow} & R_{i+1,S} & \hookrightarrow & L_{i+1,F} \\ {\scriptstyle m_{i+1,S}}\downarrow & (1) & \downarrow{\scriptstyle n_{i+1,S}} & (2) & \downarrow{\scriptstyle m_{i+1,F}} \\ G_{i0} & \underset{t_{i+1,S}}{\hookrightarrow} & G_{i+1,0} & \overset{g_{i+1}}{\hookrightarrow} & G_0 \hookrightarrow G_i \end{array}$$

□

In general, the termination of the on-the-fly construction cannot be guaranteed. But for the case that all source rules create new elements also the termination of the on-the-fly construction is ensured under the common assumption that the considered graphs and rules are finite on the graph part and the amount of rules is finite. Finiteness on the graph part means that we can still use infinite carrier sets for the algebras that are used for attribution. Thus an attributed graph G is finite on the graph part, if all sets of nodes and edges of the underlying E-graph except the set of data values V_D are finite (see Def. 3.1.5), or in a more general notion, if G has finitely many $\mathcal{M}$-subobjects (see Rem. 5.2.3 below).

Remark 5.2.3 (Finite Models)**.** *A model M_1 given by an object in an $\mathcal{M}$-adhesive category is finite, if there are finitely many $\mathcal{M}$-subobjects $M' \subseteq [M_1]$ for the equivalence class $[M_1]$ of isomorphic objects of M_1. This means for typed attributed graphs, where the class $\mathcal{M}$ is given by morphisms that are injective on the graph part and isomorphisms on the data part, that the graph component of M_1 is a finite and the data algebra for attribution can have arbitrary data carrier sets.*

Theorem 5.2.4 (Termination)**.** *Given a source model $G^S \in VL_S$ and a finite set of triple rules, such that G^S and the rule components are finite on the graph part and the triple rules are creating on the source component. Then, the on-the fly construction terminates.*

Proof. Because the source rules are creating we know that the sequence of inclusions $G^S_{00} \xhookrightarrow{t^S_{1,S}} G^S_{10} \xhookrightarrow{t^S_{2,S}} G^S_{20} \dots$ is strictly increasing. Because the graph and the rules are finite on the graph part and the rules are isomorphic on the algebra part we have that such an increasing sequence is finite. This means that we have, after a finite number of steps, that either $G_{n0} = G_0$ and the procedure terminates, or there are no more forward rules with forward consistent matches and the procedure aborts. In the case of abortion the construction may backtrack, but since we have a finite number of rules and a graph that is finite on the structural part we know that backtracking has to be performed only a finite number of times.

□

Using the equivalence of model transformations based on forward rules with those based on forward translation rules according to Thm. 5.1.34 in Sec. 5.1.2 we can directly conclude that the above results also hold for model transformations based on forward translation rules as stated by Cor. 5.2.6 below. Note that our termination criterion (source rules are creating) is sufficient and in many cases also necessary to ensure termination. Termination for model transformations based on forward translation rules means that each transformation sequence via forward translation rules can be extended only finitely many times and furthermore, if backtracking occurs then only finitely many sequences are constructed.

Definition 5.2.5 (Termination of Model Transformations Based on Forward Translation Rules)**.** *A model transformation based on forward translation rules TR_{FT} is* terminating*, if for each source model $G^S \in VL_S$ we have that each TGT-sequence starting at $G'_0 = (G^S \leftarrow \varnothing \rightarrow \varnothing)$ cannot be infinitely often extended by a further transformation step via TR_{FT}.*

Corollary 5.2.6 (Termination, Syntactical Correctness and Completeness - Forward Translation Rules)**.** *Each model transformation $MT : VL_{S0} \Rrightarrow VL_{T0}$ based on forward translation rules is*

- terminating, *if the set of forward translation rules is finite, all rule components as well as the given source model are finite on the graph part and each forward translation rule changes at least one translation attribute from "**F**" to "**T**",*
- syntactically correct, *i.e. for each model transformation sequence* $(G^S, G'_0 \overset{tr^*_{FT}}{\Longrightarrow} G'_n, G^T)$ *there is* $G \in VL$ *with* $G = (G^S \leftarrow G^C \rightarrow G^T)$ *implying further that* $G^S \in VL_S$ *and* $G^T \in VL_T$*, and it is*
- complete, *i.e. for each* $G^S \in VL_S$ *there is* $G = (G^S \leftarrow G^C \rightarrow G^T) \in VL$ *with a model transformation sequence* $(G^S, G'_0 \overset{tr^*_{FT}}{\Longrightarrow} G'_n, G^T)$*. Vice versa, for each* $G^T \in VL_T$ *there is* $G = (G^S \leftarrow G^C \rightarrow G^T) \in VL$ *with a model transformation sequence* $(G^S, G'_0 \overset{tr^*_{FT}}{\Longrightarrow} G'_n, G^T)$.

Proof. The syntactical correctness and completeness results follow directly using Thm. 5.1.34 in Sec. 5.1.34 and Thm.5.2.1 above. By Def. 5.1.28 we have that a rule changes the translation attributes iff the source rule of the original triple rule is creating, which is a sufficient criteria for termination by Thm. 5.2.4 above. □

In some cases, model transformations do not satisfy the termination criterion for Thm. 5.2.4. This is also the case for our case study in Ch. 6, because there are rules, which are not creating on the source component. For this reason, we now present an extended result for termination, which we apply in Ch. 6 for the case study to show termination. The main idea is that the source identic rules are independent of each other and contain additional NACs that prevent the rules to be applied twice at the same match.

In order to ensure independence of the forward translation rules based on critical pair analysis, we invert rules in the way that the left and right hand side is exchanged and the negative application conditions are shifted over the rule. For this purpose, the shift construction for right negative application conditions to left ones is performed using the construction according to Def. 7.16 in [EEPT06].

Definition 5.2.7 (Shift of NACs over rules)**.** *Given a rule* $p = ((L \leftarrow K \rightarrow R), N_L, N_R)$ *with a set* N_L *of left negative application conditions (over* L*) and a set* N_R *of right negative application conditions (over* R*). The transformation* $\mathrm{L}_p(n_i, N_i)$ *for each right negative application condition* $(n : R \rightarrow N, N) \in N_R$ *is given as follows. If the pushout complement* (2) *exists then* $\mathrm{L}_p(n, N) = \{(n' : L \rightarrow N')\}$*, where* (2) *is constructed as pushout complement and* (1) *as pushout. Otherwise, the result is given by* $\mathrm{L}_p(n, N) = \emptyset$.

$$\begin{array}{ccccc} L & \xleftarrow{l} & K & \xrightarrow{r} & R \\ \downarrow{\scriptstyle n'} & (1) & \downarrow & (2) & \downarrow{\scriptstyle n} \\ N' & \xleftarrow{l^*} & Z & \xrightarrow{r^*} & N \end{array}$$

The shift transformation $\mathrm{L}_p(N_R)$ *for the set of right negative application conditions is given by* $\mathrm{L}_p(N_R) = \bigcup_{(n,N)\in N_R} \mathrm{L}_p((n, N))$*. and the new set of left negative application conditions is given by* $N'_L = N_L \cup \mathrm{L}_p(N_R)$ *and the new set of right NACs is given by* $N'_R = \emptyset$.

Fact 5.2.8 (Shift of NACs over Rules)**.** *For every production p with left NACs N_L and right NACs N_R the shift transformation $\mathrm{L}_p(N_R)$ as defined in Def. 5.2.7 leads to a set of left NACs for p, such that applicability is preserved and reflected, i.e. for all direct transformations $G \overset{p,m}{\Longrightarrow} H$ with comatch n, $m \models \mathrm{L}_p(N_R) \Leftrightarrow n \models N_R$.*

Proof. The follows from Thm. 7.17 in [EEPT06], which shows the result for general application conditions, where NACs are a special kind. □

Based on Fact 5.2.8 above we now define independence of rules using the notion of critical pairs and Fact 5.2.10 below ensures that sequential independence of rules ensures implies sequential independence of subsequent transformation steps via these rules.

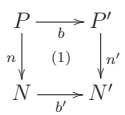
Definition 5.2.9 (Independence of Rules)**.** *Given two rules p_1 and p_2, then they are called* sequentially independent*, if there is no critical pair for (p_1^{-1}, p_2) and no critical pair for (p_1, p_2^{-1}), where $p^{-1} = ((R \overset{r}{\leftarrow} K \overset{l}{\rightarrow} L), N')$ denotes the inverted rule of the rule $p = ((L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R), N)$ and N' is obtained by shifting all NACs in N over the rule p.*

Fact 5.2.10 (Independent Rules)**.** *Given two sequentially independent rules r_1 and r_2, then any two transformation steps $G_0 \overset{r_1,m_1}{\Longrightarrow} G_1 \overset{r_2,m_2}{\Longrightarrow} G_2$ and any two transformation steps $G_0' \overset{r_2,m_2'}{\Longrightarrow} G_1' \overset{r_1,m_1'}{\Longrightarrow} G_2'$ are sequentially independent.*

Proof. According to the proof of the Local Church-Rosser Theorem (see Thm. 5.12 in [EEPT06] and Thm. 3.4.3 in [Lam09] for the case with NACs) we have that sequential independence of $G_0 \overset{r_1,m_1}{\Longrightarrow} G_1 \overset{r_2,m_2}{\Longrightarrow} G_2$ is equivalent to parallel independence of $G_0 \overset{r_1^{-1},n_1}{\Longleftarrow} G_1 \overset{r_2,m_2}{\Longrightarrow} G_2$, where n_1 is the comatch of the transformation step $G_0 \overset{r_1,m_1}{\Longrightarrow} G_1$. The analogous results holds for the steps $G_0' \overset{r_2,m_2'}{\Longrightarrow} G_1' \overset{r_1,m_1'}{\Longrightarrow} G_2'$. Thus, by completeness of critical pairs (Thm. 3.7.6 in [Lam09]) we can deduce that if the two steps $G_0 \overset{r_1,m_1}{\Longrightarrow} G_1 \overset{r_2,m_2}{\Longrightarrow} G_2$ are sequentially dependent, then there is a critical pair for (r_1^{-1}, r_2) and if the two steps $G_0' \overset{r_2,m_2'}{\Longrightarrow} G_1' \overset{r_1,m_1'}{\Longrightarrow} G_2'$ are sequentially dependent, then there is a critical pair for (r_1, r_2^{-1}). Therefore, both transformation sequences are sequentially independent. □

In order to define termination NACs that prevent a rule to be applied twice at the same match, we first define a general shift transformation of NACs over morphisms, which ensures by Fact 5.2.12 below that the applicability is preserved and reflected.

Definition 5.2.11 (Shift of NACs over Morphisms)**.** *The transformation* Shift *from morphisms $b : P \to P'$ and negative application conditions $(n : P \to N, N)$ to negative application conditions $(n' : P' \to N', N')$ is given by*



$\text{Shift}(b,(n,N)) = \{(n',N') \mid (n',b') \in \mathcal{F}\}$ *for*
$\mathcal{F} = \{(n',b') \mid (n',b')$ *jointly epimorphic,* $b' \in \mathcal{M}$, (1) *commutes*$\}$.

Fact 5.2.12 (Shift of NACs over Morphisms)**.** *The transformation* Shift *from morphisms and NACs to NACs preserves and reflects applicability, i.e. given a morphisms* $b : P \to P'$*, and a NAC* $(n : P \to N, N)$ *then for each morphism* $m : P' \to G$ *it holds that:* $m \circ b \models (n,N) \Leftrightarrow [\forall\, (n',N') \in \text{Shift}(b,(n,N)) :\ m \models (n',N')]$.

Proof. The follows from Lem. 1 in [EHL10b], which shows the result for general application conditions, where NACs are a special kind. □

Based on the shift construction for NACs in Def. 5.2.11 above we now define so-called self-disabling rules, which contain termination NACs that ensure that the rules cannot be applied twice at the same match.

Definition 5.2.13 (Self-disabling Rule)**.** *Given a rule* $p = ((L \xleftarrow{l} K \xrightarrow{r} R), N)$*, such that the shift construction* $\text{Shift}(l,(r,R))$ *(see Def. 5.2.11 yields of* p*. Furthermore, the rule contains a NAC* $(n'_1 \circ n_1)$ *for each NAC* $(n_1,N_1) \in \text{Shift}(l,(r,R))$ *and for each possible epimorphic match morphism* $n'_1 : N \to M$*, i.e. all possible foldings of* N*, where some elements are identified. Then,* p *is called* self-disabling*. The NACs in* $\text{Shift}(l,(r,R))$ *and their folded NACs are called* termination *NACs.*

Theorem 5.2.14 below generalizes the termination result for model transformations based on TGGs in Thm. 5.2.4 to the case with rules that are identities on the source component, if additional suitable conditions are satisfied. Intuitively, the rules which are not creating on the source component have to be self-disabling via termination NACs and they have to be independent from each other, if the termination NACs are not considered.

Theorem 5.2.14 (Termination with Self-Disabling Rules)**.** *A model transformation* $MT : VL_T \Rightarrow VL_T$ *via a finite set of forward translation rules* TR_FT *terminates for any finite source model* $G^S \in VL_S$*, if* TR_{FT} *can be divided into two subsets* $TR_{FT} = TR_{FT,1} \cup TR_{FT,2}$*, such that:*

- $TR_{FT,1}$ *consists of source-identic and self-disabling forward translation rules, which are pairwise independent if all NACs are removed and*
- $TR_{FT,2}$ *consists of source changing forward translation rules, i.e. each rule* $tr_{FT,i} \in TR_{FT,2}$ *changes at least one translation attribute.*

Proof. The termination criteria of Thm. 5.2.4 does not directly apply, because the model transformation may contain rules which are not creating on the source component, but they are identical. Therefore, the forward translation rules of $TR_{FT,1}$ do not modify any translation attribute. Each of the forward translation rules in $TR_{FT,2}$ changes at least one translation attribute from $\mathbf{F}$ to $\mathbf{T}$, none of them changes a translation attribute from $\mathbf{T}$ to $\mathbf{F}$ and none of them increases the amount of translation attributes. Each forward translation sequence starts with $(Att^{\mathbf{F}}(G^S) \leftarrow \varnothing \rightarrow \varnothing)$, where $(Att^{\mathbf{F}}(G^S)$ has finitely many translation attributes. Therefore, the forward translation rules of $TR_{FT,2}$ cannot be applied infinitely often in any transformation sequence starting with a finite source model, i.e. at a source model that is finite on the graph part (= result R1).

Now, consider a transformation subsequence $s = (G_i \overset{tr^*_{FT}}{\Longrightarrow} G_k)$ via forward translation rules in $TR_{FT,1}$ starting at a finite graph G_i. For each rule $tr_{FT,l}$ in $TR_{FT,1}$ there are finitely many matches into G_i. Furthermore, if NACs are neglected then all pairs of rules in $TR_{FT,1}$ are independent. So, consider the transformation sequence s' obtained from s, where all NACs of the rules are removed. Then we know by Fact 5.2.10 that all steps in s' are sequentially independent. Therefore, we can switch the steps and each step can be performed already at G_i, i.e. the matches used in s are already available at G_i. Now, since each step in s is non-deleting we further have that if the NACs of the rules are satisfied at the steps $(G_{j-1} \overset{tr_{j,FT}}{\Longrightarrow} G_j)$ in s then they are also satisfied in the subgraph $G_i \subseteq G_{j-1}$. Thus, all steps of s can be performed at G_i. Furthermore, each rule in $TR_{FT,1}$ is self-disabling. By Fact 5.2.12 this implies that each match can only be used for one application along injective matches and since we additionally have termination NACs for all foldings according to Def. 5.2.13 we derive the same result for arbitrary matches. For this reason, each rule in $TR_{FT,1}$ is applied finitely many times in s. This implies that s is finite (= result R2).

Finally there are only finitely many transformation sequences starting with $(Att^{\mathbf{F}}(G^S) \leftarrow \varnothing \rightarrow \varnothing)$, because at each intermediate step there are finitely many rules with finitely many matches. Together with (R1) and (R2) we conclude that the execution of the model transformation via forward translation rules terminates. □

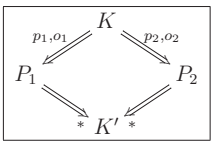
5.2.2 Functional Behaviour and Information Preservation

As shown in Sec. 5.2.1 before, syntactical correctness and completeness are guaranteed in general for model transformations based on forward rules and equivalently for those based on forward translation rules, both presented in Sec. 5.1. According to the list of functional properties that may have to be ensured for model transformations only two properties are remaining - semantic correctness and functional behaviour. In this section we will concentrate on the analysis of functional behaviour and, as a special property of bidirectional model transformations, information preservation. Semantic correctness of model

transformations, however, is a huge research field and we have started to develop analysis techniques that can handle restricted case studies (see e.g. [HHK10]).

Functional behaviour of a model transformation means that each model of the source language $\mathcal{L}_S \subseteq VL_S$ is transformed into a unique model of the target language. In many cases model transformations are desired to ensure this crucial property. The first part of this section presents new techniques especially developed to show functional behaviour of correct and complete model transformations based on TGGs. The main concepts of these techniques are presented as well in [HEOG10, HEGO10a, HEGO10b, HEGO10c] and they are based on the notion of additional *filter NACs* .

Definition 5.2.15 (Functional Behaviour of Model Transformations)**.** *A model transformation MT based on forward translation rules has* functional behaviour *if each execution of MT starting at a source model G^S of the source language $\mathcal{L}_S \subseteq VL_S$ leads to a unique target model $G^T \in VL_T$. The execution of MT requires backtracking, if there are terminating TGT-sequences $(Att^F(G^S) \leftarrow \varnothing \rightarrow \varnothing) \overset{tr^*_{FT}}{\Longrightarrow} G'_n$ with $G'^S_n \neq Att^T(G^S)$.*



The classical and general approach for showing functional behaviour of a rule based system is to verify that the system is confluent, i.e. to show that all diverging derivation paths starting at the same model finally meet again. For this purpose, it is sufficient to show termination and local confluence according to Newman's Lemma [New42]. Local confluence means that each two transformation steps $(K \overset{p_1,m_1}{\Longrightarrow} P_1)$ and $(K \overset{p_2,m_2}{\Longrightarrow} P_2)$ can be merged to a graph K' via some transformation sequences $(P_1 \overset{*}{\Longrightarrow} K')$ and $(P_2 \overset{*}{\Longrightarrow} K')$.

In order to show local confluence it is sufficient to consider only the so-called *critical pairs* $(P_1 \Leftarrow K \Rightarrow P_2)$, which specify conflicts in minimal context as shown in [EEPT06] for systems without NACs and extended to systems with NACs in [Lam09]. Minimal context means that each graph K is a subgraph of a possible overlapping of the rule components, such that the two transformation steps are parallel dependent. By completeness of critical pairs, each pair of parallel dependent transformation steps embeds a critical pair. In order to ensure that the meeting transformation sequences of the critical pair induce meeting transformation sequences for the bigger context, pure confluence of the critical pairs is not sufficient as shown by Plump [Plu93, Plu05]. For this reason the notions of strict confluence (see [EEPT06]) and NAC-strict confluence (see [Lam09]) were introduced and shown to be sufficient for systems without and with NACs, respectively. Strict confluence requires that the preserved elements of the given steps are preserved in the merging steps. Furthermore, in the presence of NACs, we also have to ensure that NAC-consistency of the merging steps for the confluence conflict is implied by the NAC-consistency of the diverging steps of the critical pair. NAC-consistency of an embedding $k : G \rightarrow G'$ for a transformation step $G \overset{p,m}{\Longrightarrow} H$ means that there is a transformation step $G' \overset{p,m'}{\Longrightarrow} H'$ with

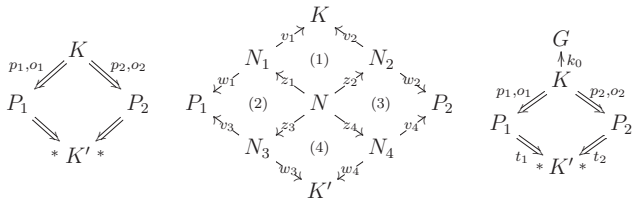


Figure 5.15: NAC-strict confluence

$m' = k \circ m$ which satisfies the NACs of p. Concerning a transformation sequence, NAC-consistency can be checked by constructing the concurrent rule of the sequence [Lam09], which combines the involved NACs in a suitable way. Let us recall the basic notions for critical pairs according to [EEPT06, Lam09].

Definition 5.2.16 (NAC-strict Confluence of Critical Pairs)**.** *A critical pair* $CP = (P_1 \xleftarrow{p_1,o_1} K \xrightarrow{p_2,o_2} P_2)$*. is called* strictly confluent*, if we have the following:*

1. Confluence*: the critical pair is confluent, i.e. there are transformations* $t_1 : P_1 \overset{*}{\Rightarrow} K'$ *and* $t_2 : P_2 \overset{*}{\Rightarrow} K'$ *with derived spans* $der(t_i) = (P_i \xleftarrow{v_{i+2}} N_{i+2} \xrightarrow{w_{i+2}} K')$ *for* $i = 1, 2$*.*

2. Strictness*: Let* $der(K \xrightarrow{p_i,o_i} P_i) = (K \xleftarrow{v_i} N_i \xrightarrow{w_i} P_i)$ *for* $i = 1, 2$*, and let* N *be the pullback object of the pullback* (1)*. Then, there are morphisms* z_3 *and* z_4 *such that* (2)*,* (3)*, and* (4) *in Fig. 5.15 commute.*

3. NAC-consistency*: For every injective morphism* $k_0 : K \to G$ *that is NAC consistent with respect to* $K \xrightarrow{p_1,o_1} P_1$ *and* $K \xrightarrow{p_2,o_2} P_2$ *in Fig. 5.15 it follows that* k_0 *is also NAC consistent with respect to* t_1 *and* t_2 *.*

However, while termination of model transformations based on forward rules respectively forward translation rules can be ensured quite easily by checking that all TGG-triple rules are creating on the source component, this is not the case for local confluence. In fact, the system of forward translation rules of our case study *CD2RDBM* is not locally confluent, but we can show in Ex. 5.2.32 that the model transformation has functional behaviour. Indeed, functional behaviour of a model transformation does not require general confluence of the underlying system of operational rules. Confluence only needs to be ensured for transformation paths which lead to completely translated models. More precisely, derivation paths leading to a point for backtracking do not influence the functional behaviour. For this reason, we introduce so-called *filter NACs* that extend the model transformation rules in order to avoid misleading paths that cause backtracking, such that

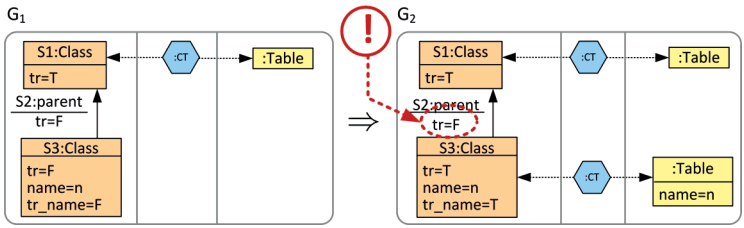


Figure 5.16: Step $G_1 \xRightarrow{Class2Table_{FT}} G_2$ with misleading graph G_2

the backtracking for the extended system is reduced substantially. By Fact 5.2.25 we ensure that the overall behaviour of the model transformation w.r.t. the model transformation relation is still preserved. As first important result we show by Thm. 5.2.27 that functional behaviour of a model transformation is ensured by confluence (termination and local confluence) of the system of forward translation rules enriched by filter NACs. Furthermore, we characterize strong functional behaviour of a terminating model transformation based on forward translation rules with filter NACs in Thm. 5.2.31 by the condition that all significant critical pairs, which are a subset of all critical pairs, are strictly confluent. Compared with functional behaviour we additionally require the uniqueness of the model transformation sequence up to switch equivalence for strong functional behaviour.

The addition of filter NACs therefore has two advantages. On the one hand, the analysis of functional behaviour is improved, because the possible conflicts between the transformation rules are reduced and we will show in this section that filter NACs allow us to verify functional behaviour for our case study *CD2RDBM*. On the other hand, filter NACs improve the efficiency of the execution by cutting off possible backtracking paths and in Sec.5.3.2 we show the power of this reduction technique based on a clear benchmark concerning the case study *CD2RDBM*. Filter NACs are based on the following notion of misleading graphs, which can be seen as model fragments that are responsible for the backtracking of a model transformation.

Definition 5.2.17 (Translatable and Misleading Graphs)**.** *A triple graph with translation attributes G is* translatable *if there is a transformation $G \overset{*}{\Rightarrow} H$ such that H is completely translated. A triple graph with translation attributes G is* misleading*, if every triple graph G' with translation attributes and $G' \supseteq G$ is not translatable.*

Example 5.2.18 (Misleading Graph)**.** *Consider the transformation step shown in Fig. 5.16. The resulting graph G_2 is misleading according to Def. 5.2.17, because the edge $S2$ is labelled with a translation attribute set to "*$\mathbf{F}$*", but there is no rule which may change this attribute in any bigger context at any later stage of the transformation. The only*

*rule which changes the translation attribute of a "parent"-edge is "$Subclass2Table_{FT}$", but it requires that the source node $S3$ is labelled with a translation attribute set to "**F**". However, forward translation rules do not modify translation attributes if they are set to "**T**" already and additionally do not change the structure of the source component.*

Definition 5.2.19 (Filter NAC)**.** *A filter NAC n for a forward translation rule $tr_{FT} : L_{FT} \to R_{FT}$ is given by a morphism $n : L_{FT} \to N$, such that there is a* TGT *step $N \overset{tr_{FT},n}{\Longrightarrow} M$ with M being misleading. The extension of tr_{FT} by some set of filter NACs is called forward translation rule tr_{FN} with filter NACs.*

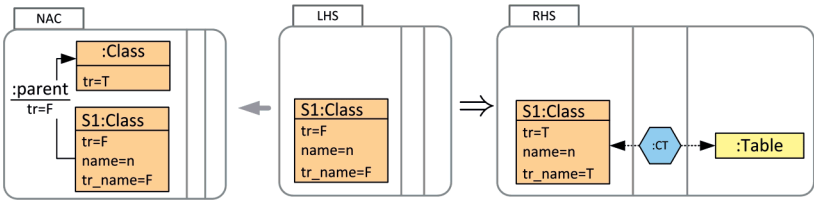


Figure 5.17: A forward translation rule with filter NAC: $Class2Table_{FN}$

Example 5.2.20 (Forward Translation Rule with Filter NACs)**.** *The rule $Class2Table_{FT}$ is extended by a filter NAC in Fig. 5.17, which is obtained from the graph G_1 of the transformation step $G_1 \overset{Class2Table_{FT}}{\Longrightarrow} G_2$ in Fig. 5.16, where G_2 is misleading according to Ex. 5.2.18. In Fact 5.2.21 below we present how such filter NACs are generated automatically.*

A direct construction of filter NACs according to Def. 5.2.19 would be inefficient, because the size of the considered graphs to be checked is unbounded. For this reason we now present efficient techniques which support the generation of filter NACs and we can bound the size without losing generality. At first we present a static technique for a subset of filter NACs and thereafter, a dynamic generation technique leading to a much larger set of filter NACs. The first procedure in Fact 5.2.21 below is based on a sufficient criterion for checking the misleading property. Concerning our example this static generation leads to the filter NAC shown in Fig. 5.17 for the rule $Class2Table_{FT}$ for an incoming edge of type "$parent$".

Fact 5.2.21 (Static Generation of Filter NACs)**.** *Given a triple graph grammar, then the following procedure applied to each triple rule $tr \in TR$ generates filter NACs for the derived forward translation rules TR_{FT} leading to forward translation rules TR_{FN} with filter NACs:*

- *Outgoing Edges: Check whether the following properties hold*
  - *tr creates a node $(x : T_x)$ in the source component and the type graph allows outgoing edges of type " T_e" for nodes of type " T_x", but tr does not create an edge $(e : T_e)$ with source node x.*
  - *Each rule in TR which creates an edge $(e : T_e)$ also creates its source node.*
  - *Extend L_{FT} to N by adding an outgoing edge $(e : T_e)$ at x together with a target node. Add a translation attribute for e with value $\mathbf{F}$. The inclusion $n : L_{FT} \to N$ is a NAC-consistent match for tr.*

  For each node x of tr fulfilling the above conditions, the filter NAC $(n : L_{FT} \to N)$ is generated for tr_{FT} leading to tr_{FN}.
- *Incoming Edges: Dual case, this time for an incoming edge $(e : T_e)$.*
- *TR_{FN} is the extension of TR_{FT} by all filter NACs constructed above.*

Proof. Consider a generated NAC $(n : L_{FT} \to N)$ for a node x in tr with an outgoing edge e in $N \setminus L$. A transformation step $N \xRightarrow{tr_{FT},n} M$ exists according to Fact 5.1.37 and leads to a graph M, where the edge e is still labelled with a translation attribute set to "**F**", but x is labelled with "**T**", because it is matched by the rule. Now, consider a graph $H' \supseteq M$, such that H' is a graph with translation attributes over a graph without translation attributes H, i.e. $H' = H \oplus Att_{H_0}$ for $H_0 \subseteq H'$ meaning that H' has at most one translation attributes for each element in H without translation attributes.

In order to have that M is misleading (Def. 5.2.17), it remains to show that H' is not translatable. Forward translation rules only modify translation attributes from "**F**" to "**T**", they do not increase the amount of translation attributes of a graph and no structural element is deleted. Thus, each graph H_i in a TGT sequence $H' \xRightarrow{tr^*_{FT}} \overline{H}_n$ will contain the edge e labelled with "**F**", because the rules, which modify the translation attribute of e are not applicable due to x being labelled with "**T**" in each graph $\overline{H}_i$ in the sequence and there is only one translation attribute for x in H'. Thus, each $\overline{H}_n$ is not completely translated and therefore, M is misleading. This means that $(n : L_{FT} \to N)$ is a filter NAC of tr_{FT}. Dualizing the proof leads to the result for a generated NAC w.r.t. an incoming edge. □

The following dynamic technique for deriving relevant filter NACs is based on the generation of critical pairs, which define conflicts of rule applications in a minimal context. By the completeness of critical pairs (Lemma 6.22 in [EEPT06]) we know that for each pair of two parallel dependent transformation steps there is a critical pair which can be embedded. For this reason, the generation of critical pairs can be used to derive filter NACs. A critical pair either directly specifies a filter NAC or a conflict that may lead to non-functional behaviour of the model transformation.

In order to provide tool support for the dynamic generation of filter NACs and the analysis of functional behaviour of model transformations we apply the flattening construction for triple graphs, which we presented in [EEH08c, EEH08d], and derive a "plain" graph grammar GG. The analysis of GG can be performed using the implemented critical pair analysis of the tool AGG [AGG10] for typed attributed graph grammars which allows the designer to generate and analyse all critical pairs of a grammar. In order to apply the flattening construction we additionally require that the correspondence component TG^C of the type graph TG is discrete, i.e. has no edges. This condition is fulfilled for our case study and many others as well. An extension of the tool AGG to general triple graphs will be part of future work.

The flattening of a triple graph $G = (G^S \xleftarrow{s_G} G^C \xrightarrow{t_G} G^T)$ is a (single plain) graph $\mathcal{F}(G)$ obtained by disjoint union of the components G^S, G^C and G^T extended by additional edges $Link_S$ and $Link_T$, which encode the internal morphisms s_G and t_G. Moreover, the flattened type graph $\mathcal{F}(TG)$ contains new edge as well and they define the edge types for the additional edges that occur in the flattened triple graphs $\mathcal{F}(G)$. In the examples we will denote the additional edge type equally by the suffix ":morph".

Definition 5.2.22 (Flattening Construction)**.** *Let $G = (G^S \xleftarrow{s_G} G^C \xrightarrow{t_G} G^T)$ be a triple graph. The* flattening *$\mathcal{F}(G)$ of G is a plain graph defined by the disjoint union $\mathcal{F}(G) = G^S + G^C + G^T + Link_S(G) + Link_T(G)$ with links (additional edges) defined by*
$Link_S(G) = \{(x, y) \mid x \in V_{G^C}, y \in V_{G^S}, s_G(x) = y\}$,
$Link_T(G) = \{(x, y) \mid x \in V_{G^C}, y \in V_{G^T}, t_G(x) = y\}$
with $src_{\mathcal{F}(G)}((x, y)) = x$ and $tgt_{\mathcal{F}(G)}((x, y)) = y$ for $(x, y) \in Link_S \cup Link_T$.

As shown by Thm. 2 in [EEH08c] there is a one-to-one correspondence between a triple graph transformation sequences and its flattened plain transformation sequence starting at the flattened start graph and applying the flattened triple rules. Hence, we can analyse confluence, in particular critical pairs, of a set of triple rules by analysing the corresponding set of flattened rules. This allows us to flatten the forward translation rules of a model transformation and generate the critical pairs in AGG in order to analyse the functional behaviour of the model transformation.

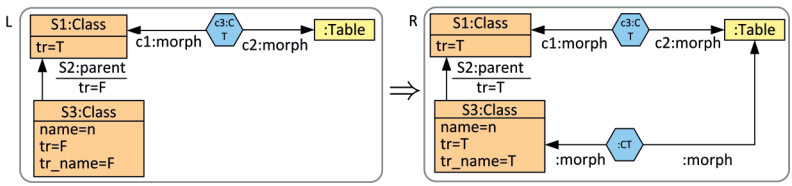


Figure 5.18: Flattening of the forward translation rule $Subclass2Table_{FT}$

Example 5.2.23 (Flattened Forward Translation Rule)**.** *Figure 5.18 shows the result of the flattening construction applied to the forward translation rule $Subclass2Table_{FT}$, which is depicted in the right part of Fig. 5.12. The triple graphs are flattened to plain graphs, where each mapping of the internal graph morphisms of the triple graphs is encoded as an explicit edge of type morph denoted by a solid line.*

Based on the flattening construction we derive an equivalent plain graph transformation system from the system of forward translation rules. This allows us to use the critical pair generation and analysis of the tool AGG [AGG10] for the dynamic generation of filter NACs. A critical pair $P_1 \xleftarrow{tr_{1,FT}} K \xrightarrow{tr_{2,FT}} P_2$ consists of a pair of parallel dependent transformation steps. If a critical pair contains a misleading graph P_1, we can use the overlapping graph K as a filter NAC of the rule $tr_{1,FT}$. However, checking the misleading property needs human assistance, such that the generated critical pairs can be seen as filter NAC candidates. But we are currently working on a technique that uses a sufficient criteria to check the misleading property automatically, and we are confident that this approach will provide a powerful generation technique.

Fact 5.2.24 (Dynamic Generation of Filter NACs)**.** *Given a set of forward translation rules, then generate the set of critical pairs $P_1 \xleftarrow{tr_{1,FT},m_1} K \xrightarrow{tr_{2,FT},m_2} P_2$. If P_1 (or similarly P_2) is misleading, we generate a new filter NAC $m_1 : L_{1,FT} \to K$ for $tr_{1,FT}$ leading to $tr_{1,FN}$, such that $K \xrightarrow{tr_{1,FN},m_1} P_1$ violates the filter NAC. Hence, the critical pair for $tr_{1,FT}$ and $tr_{2,FT}$ is no longer a critical pair for $tr_{1,FN}$ and $tr_{2,FT}$. But this construction may lead to new critical pairs for the forward translation rules with filter NACs. The procedure is repeated until no further filter NAC can be found or validated. This construction starting with TR_{FT} always terminates if the structural part of each graph of a rule is finite.*

Proof. The constructed NACs are filter NACs, because the transformation step $K \xrightarrow{tr_{1,FT},m_1} P_1$ contains the misleading graph P_1. The procedure terminates, because the critical pairs are bounded by the amount of possible pairwise overlappings of the left hand sides of the rules. The amount of overlappings can be bounded by considering only constants and variables as possible attribute values. □

For our case study the dynamic generation terminates already after the second round, which is typical for practical applications, because the amount of already translated elements in the new critical pairs usually decreases. Furthermore, the amount of NACs can be reduced by combining similar NACs differing only on some translation attributes. The remaining critical pairs that do not specify filter NACs show effective conflicts between transformation rules, and they can be provided to the developer of the model transformation to support the design phase.

As shown by Fact 5.2.25 below, filter NACs do not change the behaviour of model transformations. The only effect is that they filter out derivation paths, which would lead

to misleading graphs, i.e. to backtracking for the computation of the model transformation sequence. This means that the filter NACs filter out backtracking paths. This equivalence is used on the one hand for the analysis of functional behaviour in Thms. 5.2.27 and 5.2.31 and furthermore, for improving the efficiency of the execution of model transformations as explained in Sec. 5.3.

Fact 5.2.25 (Equivalence of Transformations with Filter NACs)**.** *Given a triple graph grammar $TGG = (TG, \varnothing, TR)$ and a triple graph $G_0 = (G^S \leftarrow \varnothing \rightarrow \varnothing)$ typed over TG. Let $G'_0 = (Att^{\mathbf{F}}(G^S) \leftarrow \varnothing \rightarrow \varnothing)$, then the following are equivalent for almost injective matches:*

1. $\exists$ *a complete* TGT*-sequence* $G'_0 \overset{tr^*_{FT}, m^*_{FT}}{\Longrightarrow} G'$ *via forward translation rules.*
2. $\exists$ *a complete* TGT*-sequence* $G'_0 \overset{tr^*_{FN}, m^*_{FT}}{\Longrightarrow} G'$ *via forward translation rules with filter NACs.*

Proof Idea. Sequence 1 consists of the same derivation diagrams as Sequence 2. The additional filter NACs in sequence 2 prevent a transformation rule to create a misleading graph. Both sequences lead to completely translated models, such that we know that the matches in sequence 1 also fulfil the filter NACs of the rules in sequence 2. The full proof is given in [HEGO10c]. □

In order to analyse functional behaviour we generate the critical pairs for the system of forward translation rules and show by Thm. 5.2.27 that strict confluence of "significant" critical pairs ensures functional behaviour. A critical pair is significant if it can be embedded into two transformation sequences via forward translation rules that start at the same source model G^S, which belongs to the source language VL_S.

Definition 5.2.26 (Significant Critical Pair)**.** *A critical pair $(P_1 \overset{tr_{1,FN}}{\Longleftarrow} K \overset{tr_{2,FN}}{\Longrightarrow} P_2)$ for TR_{FN} is called significant if it can be embedded into a parallel dependent pair $(G'_1 \overset{tr_{1,FN}}{\Longleftarrow} G' \overset{tr_{2,FN}}{\Longrightarrow} G'_2)$ such that there is $G^S \in \mathcal{L}_S \subseteq VL_S$ for the language of source models $\mathcal{L}_S$ and $G'_0 \overset{tr^*_{FN}}{\Longrightarrow} G'$ with $G'_0 = (Att^{\mathbf{F}}(G^S) \leftarrow \varnothing \rightarrow \varnothing)$.*

$$G'_0 \overset{*}{\Longrightarrow} G' \overset{tr_{1,FN}}{\Longrightarrow} G1' \quad G' \overset{tr_{2,FN}}{\Longrightarrow} G'_2$$

Theorem 5.2.27 (Functional Behaviour)**.** *Let MTbe a model transformation based on forward translation rules TR_{FT} and let TR_{FN} extend TR_{FT} with filter NACs such that TR_{FN} is terminating and all significant critical pairs are strictly confluent. Then, MT has functional behaviour. Moreover, the model transformation MT' based on TR_{FN} does not require backtracking and defines the same model transformation relation, i.e. $MTR' = MTR$.*

Remark 5.2.28. *TR_{FN} is terminating if TR_{FT} is terminating. A sufficient condition for termination of TR_{FT} is given in Cor. 5.2.6. Termination of TR_{FN} with strict confluence of critical pairs implies unique normal forms by the Local Confluence Theorem in [Lam09].*

Proof. For functional behaviour of the model transformation we have to show that each source model $G^S \in VL_S$ is transformed into a unique (up to isomorphism) completely translated target model G^T, which means that there is a completely translated triple model G' with $G'^T = G^T$, and furthermore $G^T \in VL_T$.

For $G^S \in \mathcal{L}_S \subseteq VL_S$ we have by definition of VL that there is a $G^T \in VL_T$ and a TGT-sequence $\varnothing \overset{tr^*}{\Longrightarrow} (G^S \leftarrow G^C \rightarrow G^T)$ via TR and using the decomposition theorem with NACs in [EHS09a] we obtain a match consistent TGT-sequence $\varnothing \overset{tr_S^*}{\Longrightarrow} G_0 = (G^S \leftarrow \varnothing \rightarrow \varnothing) \overset{tr_F^*}{\Longrightarrow} (G^S \leftarrow G^C \rightarrow G^T)$ and by Thm. 5.1.34 a complete TGT-sequence $G'_0 = (Att^{\mathbf{F}}(G^S) \leftarrow \varnothing \rightarrow \varnothing) \overset{tr_{FT}^*}{\Longrightarrow} (Att^{\mathbf{T}}(G^S) \leftarrow G^C \rightarrow G^T) = G'$.

This means that $(G^S, G'_0 \overset{tr_{FT}^*}{\Longrightarrow} G', G^T)$ is a model transformation sequence based on TR_{FT}. Assume that we also have a complete forward translation sequence $G'_0 = (Att^{\mathbf{F}}(G^S) \leftarrow \varnothing \rightarrow \varnothing) \overset{\overline{tr}_{FT}^*}{\Longrightarrow} (Att^{\mathbf{T}}(G^S) \leftarrow \overline{G^C} \rightarrow \overline{G^T}) = \overline{G}'$. By Fact. 5.2.25 we also have the complete TGT-sequences $(G^S, G'_0 \overset{tr_{FN}^*}{\Longrightarrow} G', G^T)$ and $G'_0 \overset{tr_{FN}^*}{\Longrightarrow} G'$ and $G'_0 \overset{tr_{FN}^*}{\Longrightarrow} \overline{G}'$. Using the precondition that TR_{FN} is terminating and all significant critical pairs are strictly confluent we show that all diverging transformation sequences can be merged again. Consider the possible transformation sequences starting at G'_0 (which form a graph of transformation steps). If two diverging steps $(G'_{i+1} \overset{p_1,m_1}{\Longleftarrow} G'_i \overset{p_2,m_2}{\Longrightarrow} G''_{i+1})$. If they are parallel independent, we can apply the local Church Rosser Thm (LCR) [Lam09] and derive the merging steps $(G'_{i+1} \overset{p_2,m'_2}{\Longrightarrow} H \overset{p_1,m'_1}{\Longleftarrow} G''_{i+1})$. If they are parallel dependent diverging steps we know by completeness of critical pairs (see Thm. 3.7.6 in [Lam09]) that there is a critical pair and by Def. 5.2.26 we know that this pair is significant, because we consider transformations sequences starting at G'_0. This pair is strictly confluent by precondition. Therefore, these steps can be merged again. Now, any new diverging situation can be merged by either LCR for parallel independet steps or by strict confluence of the critical pair for the parallel dependent steps. By precondition the system is terminating. In combination, this implies that $G' \cong \overline{G}'$ and hence, $G^T \cong \overline{G}^T$.

Backtracking is not required, because termination of TR_{FN} with strict confluence of significant critical pairs implies unique normal forms by as shown above. Therefore, any terminating TGT-sequence $(Att^F(G^S) \leftarrow \varnothing \rightarrow \varnothing) \overset{tr_{FN}^*}{\Longrightarrow} G'_n$ leads to a unique G'_n up to isomorphism and by correctness and completeness (Cor. 5.2.6) we have that $G'^S_n = Att^T(G^S)$.

The model transformation relation is the same, because we have by Fact 5.2.25 the equivalence of the model transformation sequences of MT and MT'.

□

If the set of generated critical pairs of a system of forward translation rules with filter NACs TR_{FN} is empty, we can directly conclude from Thm. 5.2.27 that the corresponding system with forward translation rules TR_{FT} has functional behaviour. From an efficiency point of view, model transformations should be based on a compact set of rules, because large rule sets usually involve more attempts of matching until finding a valid match. In the optimal case, the rule set ensures that each transformation sequence of the model transformation is itself unique up to switch equivalence. For this reason, we introduce the notion of strong functional behaviour.

Definition 5.2.29 (Strong Functional Behaviour of Model Transformations)**.** *A model transformation based on forward translation rules TR_{FN} with filter NACs has* strong functional behaviour *if for each $G^S \in \mathcal{L}_S \subseteq VL_S$ there is a $G^T \in VL_T$ and a model transformation sequence $(G^S, G'_0 \overset{tr^*_{FN}}{\Longrightarrow} G'_n, G^T)$ based on forward translation rules, and moreover,*

- *any partial TGT-sequence $G'_0 \overset{tr^{i,*}_{FN}}{\Longrightarrow} G'_i$ terminates, i.e. there are finitely many extended sequences $G'_0 \overset{tr^{i,*}_{FN}}{\Longrightarrow} G'_i \overset{tr^{j,*}_{FN}}{\Longrightarrow} G'_j$, and*
- *each two TGT-sequences $G'_0 \overset{tr^*_{FN}}{\Longrightarrow} G'_n$ and $G'_0 \overset{\overline{tr}^*_{FN}}{\Longrightarrow} \overline{G}'_m$ with completely translated graphs G'_n and $\overline{G}'_m$ are switch-equivalent up to isomorphism.*

Remark 5.2.30 (Strong Functional Behaviour)**.**

1. *The sequences are terminating means that no rule in TR_{FN} is applicable any more. However, it is not required that the sequences are complete, i.e. that G'_n and $\overline{G}'_m$ are completely translated.*
2. *Strong functional behaviour implies functional behaviour, because G'_n and $\overline{G}'_m$ completely translated implies that $G'_0 \overset{tr^*_{FN}}{\Longrightarrow} G'_n$ and $G'_0 \overset{\overline{tr}^*_{FN}}{\Longrightarrow} \overline{G}'_m$ are terminating TGT-sequences.*
3. *Two sequences $t1 : G_0 \Rightarrow^* G_1$ and $t2 : G_0 \Rightarrow^* G_2$ are called switch-equivalent, written $t1 \approx t2$, if $G_1 = G_2$ and $t2$ can be obtained from $t1$ by switching sequential independent steps according to the Local Church Rosser Theorem with NACs [Lam09]. The sequences $t1$ and $t2$ are called switch-equivalent up to isomorphism if $t1 : G_0 \Rightarrow^* G_1$ has an isomorphic sequence $t1' : G_0 \Rightarrow^* G_2$ (using the same sequence of rules) with $i : G_1 \xrightarrow{\sim} G_2$, written $trace(t1') = i \circ trace(t1)$, such that $t1' \approx t2$. This means especially that the rule sequence in $t2$ is a permutation of that in $t1$.*

The third main result of this paper shows that strong functional behaviour of model transformations based on forward translation rules with filter NACs can be completely characterized by the absence of significant critical pairs.

Theorem 5.2.31 (Strong Functional Behaviour)**.** *A model transformation based on terminating forward translation rules TR_{FN} with filter NACs has strong functional behaviour and does not require backtracking iff TR_{FN} has no significant critical pair.*

Proof.

Direction "⇐": Assume that TR_{FN} has no significant critical pair. Similar to the proof of Thm. 5.2.27 we obtain for each $G^S \in VL_S$ a $G^T \in VL_T$ and a complete TGT-sequence $G'_0 \overset{tr^*_{FT}}{\Longrightarrow} G'$ and a model transformation $(G^S, G'_0 \overset{tr^*_{FT}}{\Longrightarrow} G', G^T)$ based on TR_{FT} underlying TR_{FN}. By Fact. 5.2.25 we also have a complete TGT-sequence $G'_0 \overset{tr^*_{FN}}{\Longrightarrow} G'$ and hence, also a model transformation $(G^S, G'_0 \overset{tr^*_{FT}}{\Longrightarrow} G', G^T)$ based on TR_{FT} underlying TR_{FN}. In order to show strong functional behaviour let $G'_0 \overset{tr^*_{FN}}{\Longrightarrow} G'_n$ and $G'_0 \overset{\overline{tr}^*_{FN}}{\Longrightarrow} \overline{G}'_m$ be two terminating TGT-sequences with $m, n \geq 1$. We have to show that they are switch-equivalent up to isomorphism. We show by induction on the combined length $n + m$ that both sequences can be extended to switch-equivalent sequences.

For $n + m = 2$ we have $n = m = 1$ with $t1 : G'_0 \overset{tr_{FN}, m}{\Longrightarrow} G'_1$ and $\overline{t1} : G'_0 \overset{\overline{tr}_{FN}, \overline{m}}{\Longrightarrow} \overline{G}'_1$. If $tr_{FN} = \overline{tr}_{FN}$ and $m = \overline{m}$, then both are isomorphic with isomorphism $i : \overline{G}'_1 \xrightarrow{\sim} G'_1$, such that $t1 \approx i \circ \overline{t1}$. If not, then $t1$ and $\overline{t1}$ are parallel independent, because otherwise we would have a significant critical pair by completeness of critical pairs in [Lam09]. By the Local Church Rosser Theorem [Lam09] we have $t2 : G'_1 \overset{\overline{tr}_{FN}}{\Longrightarrow} G'_2$ and $\overline{t2} : \overline{G}'_1 \overset{tr_{FN}}{\Longrightarrow} G'_2$, such that $t2 \circ t1 \approx \overline{t2} \circ \overline{t1} : G'_0 \Rightarrow^* G'_2$.

Now assume that for $t1 : G'_0 \Rightarrow^* G'_{n-1}$ and $\overline{t1} : G'_0 \Rightarrow^* \overline{G}'_m$ we have extensions $t2 : G'_{n-1} \Rightarrow^* H$, $\overline{t2} : \overline{G}'_m \Rightarrow^* H$, such that $t2 \circ t1 \approx \overline{t2} \circ \overline{t1}$.

$$\begin{array}{ccccc} G'_0 & \overset{t1}{\Longrightarrow}{}^* & G'_{n-1} & \overset{t}{\Longrightarrow} & G'_n \\ {\scriptstyle \overline{t1}}\Downarrow{\scriptstyle *} & & \Downarrow{\scriptstyle t2}{\scriptstyle *} & & \Downarrow{\scriptstyle t3}{\scriptstyle *} \\ \overline{G}'_m & \underset{\overline{t2}}{\Longrightarrow}{}^* & H & \underset{\overline{t3}}{\Longrightarrow}{}^* & K \end{array}$$

Now consider a step $t : G'_{n-1} \Rightarrow G'_n$, then we have to show that $t \circ t1$ and $\overline{t1}$ can be extended to switch-equivalent sequences. By induction hypothesis and definition of significant critical pairs also t and $t2$ can be extended by $t3 : G'_n \Rightarrow^* K$, $\overline{t3} : H \Rightarrow^* K$, such that $t3 \circ t \approx \overline{t3} \circ t2$. Now, composition closure of switch equivalence implies $t3 \circ t \circ t1 \approx \overline{t3} \circ \overline{t2} \circ \overline{t1} : G'_0 \Rightarrow^* K$. This completes the induction proof.

Now, we use that G'_n and $\overline{G}'_m$ are both terminal which implies that $t3$ and $\overline{t3} \circ \overline{t2}$ must be isomorphisms. This shows that $G'_0 \overset{tr^*_{FN}}{\Longrightarrow} G'_n$ and $G'_0 \overset{\overline{tr}^*_{FN}}{\Longrightarrow} \overline{G}'_m$ are switch-equivalent up to isomorphism.

Direction "⇒": Assume now that TR_{FN} has strong functional behaviour and that TR_{FN} has a significant critical pair. We have to show a contradiction in this case.

Let $P_1 \xleftarrow{tr_{1,FN}} K \xrightarrow{tr_{2,FN}} P_2$ be the significant critical pair which can be embedded into a parallel dependent pair $G_1 \xleftarrow{tr_{1,FN}} G' \xrightarrow{tr_{2,FN}} G_2$, such that there is $G^S \in VL_S$ with $G'_0 \xrightarrow{tr^*_{FN}} G'$ and $G'_0 = (Att^{\mathbf{F}}(G^S) \leftarrow \varnothing \rightarrow \varnothing)$. Since TR_{FN} is terminating we have terminating sequences $G_1 \Rightarrow^* G_{1n}$ via TR_{FN} and $G_2 \Rightarrow^* G_{2m}$ via TR_{FN}. By composition we have the following terminating TGT-sequences

1. $G'_0 \xrightarrow{tr_{FN}} G' \xrightarrow{tr_{1,FN}} G_1 \Rightarrow^* G_{1n}$
2. $G'_0 \xrightarrow{tr_{FN}} G' \xrightarrow{tr_{2,FN}} G_2 \Rightarrow^* G_{2m}$

Since TR_{FN} has strong functional behaviour both are switch-equivalent up to isomorphism. For simplicity assume $G_{1n} = G_{2m}$ instead of $G_{1n} \cong G_{2m}$. This implies that $n = m$ and $G' \xrightarrow{tr_{1,FN}} G_1 \Rightarrow^* G_{1n}$ is switch-equivalent to $G' \xrightarrow{tr_{2,FN}} G_2 \Rightarrow^* G_{1n}$. This means $tr_{2,FN}$ occurs in $G_1 \Rightarrow^* G_{1n}$ and can be shifted in $G' \xrightarrow{tr_{1,FN}} G_1 \Rightarrow^* G_{1n}$, such that we obtain $G' \xrightarrow{tr_{2,FN}} G_2 \Rightarrow^* G_{1n}$.

But this implies that in an intermediate step we can apply the parallel rule $tr_{1,FN} + tr_{2,FN}$ leading to parallel independence of $G' \xrightarrow{tr_{1,FN}} G_1$ and $G' \xrightarrow{tr_{2,FN}} G_2$, which is a contradiction. Hence, TR_{FN} has no significant critical pair.

It remains to show that strong functional behaviour implies that backtracking is not required. This is a direct consequence of Thm. 5.2.27, since we have no significant critical pair and therefore, all of them are strictly confluent.

□

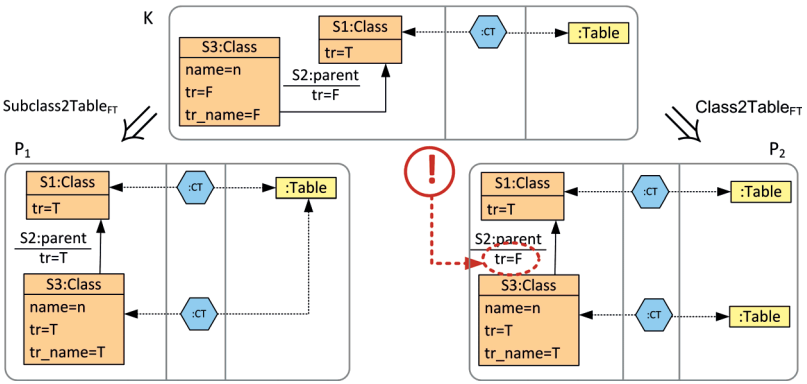


Figure 5.19: Critical pair for the rules $Subclass2Table_{FT}$ and $Class2Table_{FT}$

Example 5.2.32 (Functional Behaviour)**.** *We analyse functional behaviour of the model transformation $CD2RDBM$. First of all, $CD2RDBM$ is terminating according to Cor. 5.2.6, because all TGG-triple rules are creating in the source component as well as finite on the graph part and thus, the possible source models are finite on the graph part. For analysing local confluence we use the tool AGG [AGG10] for the generation of critical pairs. The set of derived forward translation rules from the rules TR in Figs. 5.4 and 5.5 is given by $TR_{FT} = \{Class2Table_{FT}, Subclass2Table_{FT}, Attr2Column_{FT}, Class2TPrimaryAttr2Column_{FT}, Association2ForeignKey_{FT}\}$. We exchange the forward translation rule $Class2Table_{FT}$ by the extended rule with filter NACs $Class2Table_{FN}$ as shown in Fig. 5.17, and additionally extend it by a further filter NAC obtained by the static generation according to Fact 5.2.21. AGG detects two critical pairs showing a conflict of the rule "$PrimaryAttr2Column$" with itself for an overlapping graph with two primary attributes. Both critical pairs lead to additional filter NACs by the dynamic generation of filter NACs in Fact 5.2.24 leading to a system of forward translation rules with filter NACs without any critical pairs. Thus, we can apply Thm. 5.2.31 and show that the model transformation based on the forward translation rules with filter NACs TR_{FN} has* strong functional behaviour *and does not require backtracking. Furthermore, by Thm. 5.2.27 we can conclude that the model transformation based on the forward translation rules TR_{FT} without filter NACs has* functional behaviour *and does not require backtracking. As an example, Fig. 5.10 shows the resulting triple graph (translation attributes are omitted) of a model transformation starting with the class diagram G^S.*

While functional behaviour of a model transformation ensures unique results, we are faced with a further challenge concerning bidirectional model transformations - the analysis whether and to which extent a source model can be reconstructed from the computed target model. For example, a transformation of a domain-specific model to some formal model for the purpose of validation should be reversible to transform back analysis results stemming from the formal model, and the derived new source model should not differ too much from the given one. Reversible model transformations also play an important role in the presence of system evolution. Having usually a variety of different models around in the engineering process, the evolution of one model depends on the evolution of other models. To keep models coherent to each other, model transformations have to be reversible and to a certain extent information-preserving in the sense defined below.

Hence we now present techniques for analysing model transformations based on TGGs with respect to information-preservation and complete information preservation. Interestingly, it turns out that complete information preservation, i.e. the complete reconstruction of the source model, is ensured by functional behaviour of the backward model transformation. We present the techniques for model transformations based on forward rules, but according to the equivalence result in Thm. 5.1.34, we also know that these techniques

provide the same results for model transformations based on forward translation rules with and without filter NACs.

Definition 5.2.33 (Information Preserving Model Transformation)**.** *A model transformation based on forward rules is* backward information preserving*, if for each forward model transformation sequence* $(G^S, G_0 \overset{tr_F^*}{\Longrightarrow} G_n, G^T)$ *there is a backward model transformation sequence* $(G^T, G'_0 \overset{tr'^*_B}{\Longrightarrow} G'_m, G'^S)$ *with* $G^S = G'^S$*, i.e. the source model* G^S *can be recreated from the resulting target model* G^T *via a target consistent backward transformation sequence. For backward transformations the term forward information preserving is defined dually.*

The condition under which we obtain standard backward information preservation is source consistency, i.e. model transformations based on forward rules and model transformations based on forward translation rules are backward information preserving in general as stated by Thm. 5.2.34 below.

Theorem 5.2.34 (Information Preserving Model Transformation)**.** *Each model transformation based on forward rules is backward information preserving.*

Proof. By Thm. 5.2.1 we know that the forward sequence $G_0 \overset{tr_F^*}{\Longrightarrow} G_n$ is source consistent, and by Thm. 5.1.14 we derive the target consistent backward transformation $G'_0 = (G^T \leftarrow \varnothing \rightarrow \varnothing) \overset{tr_B^*}{\Longrightarrow} G_n$ with $G_n^S = G^S$. This means that we have a backward model transformation sequence $(G^T, G'_0 \overset{tr_B^*}{\Longrightarrow} G_n, G'^S)$ with $G^S = G'^S$. □

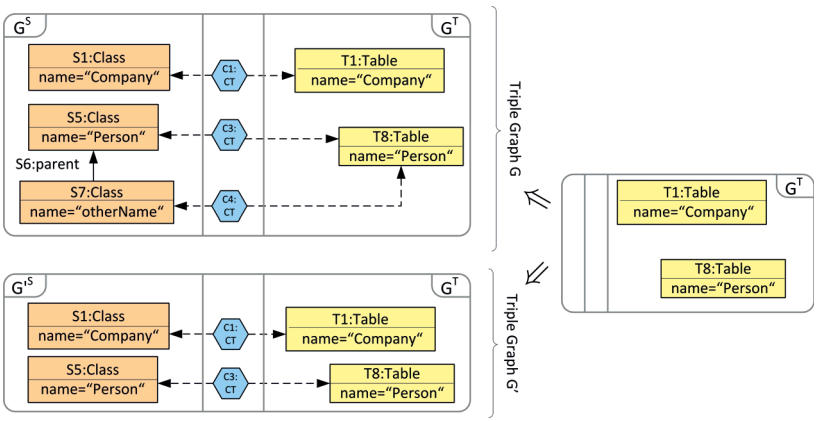


Figure 5.20: Two possible target consistent backward transformations

Example 5.2.35 (backward information preserving model transformation *CD2RDBM*)**.** *The model transformation CD2RDBM is information preserving, because it consists of model transformation sequences based on forward rules, which ensure source consistency of the forward sequences by definition. Therefore, the presented source model G^S of the triple graph in Fig. 5.10 can be reconstructed by a target consistent backward transformation sequence starting at the model $G'_0 = (\varnothing \leftarrow \varnothing \rightarrow G^T)$. But there are several possible target consistent backward transformation sequences starting at G'_0. The reason is that the rule $Subclass2Table_B$ can be applied arbitrarily often without having an influence concerning the target consistency, because the rule is identical on the target component. This means that the inheritance information within a class diagram has no explicit counterpart within a relational data base model.*

Two possible target consistent backward transformation sequences for the same derived target model G^T are presented in Fig. 5.20. The source model G^S can be transformed into $G = (G^S \leftarrow G^C \rightarrow G^T)$. But starting with G^T, both depicted backward transformation sequences are possible and target consistent. The resulting source graphs G^S and G'^S, however, differ with respect to the class node $S7$ and the edge $S6$ in G^S. Hence, not all information of G^S can be reconstructed uniquely and therefore, i.e. some information gets lost in the target model G^T.

According to Thm. 5.2.34 each model transformation based on forward rules as presented in Sec. 5.1.1 is backward information preserving. But the reconstruction of a corresponding source model from a derived target model is in general not unique. In order to ensure uniqueness of the reconstruction we now present the notion of complete information preservation. This stronger notion ensures that all information contained in a source model can be reconstructed from the derived target model itself. More precisely, starting with the target model, each backward model transformation sequence will produce the original source model. This means in particular that only one backward model transformation sequence has to be constructed. Hence, from an intuitive point of view, all information that were present in the given source model is completely preserved during the forward model transformation and stored within the constructed target model.

Definition 5.2.36 (Complete Information Preservation)**.** *A model transformation is* completely information preserving *if it is backward information preserving and furthermore, given a source model $G^S \in \mathcal{L}_S$ and the resulting target model G^T of a forward model transformation sequence, then each partial backward transformation sequence starting with G^T using the on-the-fly construction (Thm. 5.1.22) terminates and produces the given source model G^S as result.*

As the definition of complete information preservation already states, we can verify that a model transformation satisfies this property by showing that each backward transformation sequence starting at a derived target model produces unique results. Thus, we can verify

complete information preservation by showing functional behaviour of the corresponding backward model transformation with respect to the derived target models $MT(\mathcal{L}_S) \subseteq VL_T$. According to the definition of functional behaviour (Def. 5.2.15) applied to the backward model transformation, we have to concern the target DSL $\mathcal{L}'_T = MT(\mathcal{L}_S)$.

Theorem 5.2.37 (Completely Information Preserving Model Transformation)**.** *Given a model transformation MT based on forward rules. Then, MT is completely information preserving if the corresponding backward model transformation according to Def. 5.1.15 has functional behaviour with respect to the target language $\mathcal{L}'_T = MT(\mathcal{L}_S)$.*

Proof. By Thm. 5.2.34 we know that MT is backward information preserving. Now, given a model transformation sequence $(G^S, G_0 \overset{tr^*_F}{\Longrightarrow} G_n, G^T)$, we additionally know that $G^T \in VL_T$ by Thm. 5.2.1, and furthermore, we know that $G^T \in \mathcal{L}'_T = MT(\mathcal{L}_S)$. Using the functional behaviour of the corresponding backward model transformation according to Def. 5.2.15 for the language $\mathcal{L}'_T$ we know that for each model H^T the backward model transformation yields a unique $H^S \in VL_S$. Therefore, each backward model transformation sequence $(G^T, G'_0 \overset{tr^*_B}{\Longrightarrow} G'_n, G'^S)$ leads to a unique $G'^S \in VL_S$. Furthermore, we know by Thm. 5.2.34 that there is a backward model transformation sequence $(G^T, G''_0 \overset{tr^*_B}{\Longrightarrow} G''_n, G^S)$ implying $G^S \cong G'^S$, i.e. the model transformation is completely information preserving. □

Remark 5.2.38 (Complete Information Preservation)**.**

1. *As stated in Sec. 5.1.1 we generally assume that $\mathcal{L}_S \subseteq VL_S$. In our case study we have the special case $\mathcal{L}_S = VL_S$ and $\mathcal{L}_T = VL_T$. In the general case, the condition $\mathcal{L}_S \subseteq VL_S$ has to be shown for a concrete DSL $\mathcal{L}_S$ separately.*

2. *If we additionally have the case that $\mathcal{L}_S = VL_S$, then the given condition for complete information preservation is not only sufficient but also necessary. The reason is that the required unique resulting source model of the model transformation directly corresponds in this case with the definition of functional behaviour by replacing $\mathcal{L}'_T$ with VL_T.*

Example 5.2.39 (Complete Information Preservation)**.** *The model transformation $MT_1 =$ CD2RDBM is not completely information preserving. Consider e.g. the source model G^S in Fig. 5.20 of Ex. 5.2.35, where two backward model transformation sequences are possible starting with the same derived target model G^T. This means that the backward model transformation has no functional behaviour with respect to $MT_1(\mathcal{L}_S) = MT(VL_S) = VL_T = \mathcal{L}_T$.*

However, we can also consider the inverse model transformation, i.e. swapping the forward and backward direction leading to the model transformation $MT_2 =$ RDBM2CD

from relational data base models to class diagrams. In this case, the model transformation is completely information preserving meaning that each relational data base model M_{DB} can be transformed into a class diagram M_{CD}, and each data based model model M_{DB} can be completely and uniquely reconstructed from its derived class diagram M_{CD}. In other words, each class diagram resulting from a model transformation sequence of RDBM2CD contains all information that were present in the given data base model. According to Ex. 5.2.32 we know that the model transformation CD2RDBM has functional behaviour and hence, the backward model transformation of RDBM2CD has functional behaviour with respect to VL_T being equal to the source language VL_S of CD2RDBM. For this reason, we can apply Thm. 5.2.37 and have that RDBM2CD is completely information preserving. In particular, foreign keys are completely represented by associations, and primary keys by primary attributes. There is no structure within the data base model which is not explicitly represented within the class diagram.

In some cases, a model transformation should also ensure that the derived target models are included in the considered target DSL $\mathcal{L}_T$. This gives rise to the stricter notion of syntactical correctness, called strong syntactical correctness, which furthermore provides part of a sufficient condition for complete information preservation according to Cor. 5.2.41 below.

Definition 5.2.40 (Strong Syntactical Correctness)**.** *Given the source DSL $\mathcal{L}_S$ and the target DSL $\mathcal{L}_T$ of a model transformation MT. Then, MT is called* strongly syntactically correct *if it is syntactically correct and furthermore, given a model transformation sequence $(G^S, G_0 \overset{tr_F^*}{\Longrightarrow} G_n, G^T)$ with $G^S \in \mathcal{L}_S$, then $G^T \in \mathcal{L}_T$. This means that the set of derived target models belongs to the target DSL, i.e. $MT(\mathcal{L}_S) \subseteq \mathcal{L}_T$*

Corollary 5.2.41 (Completely Information Preserving Model Transformation)**.** *Given a strongly syntactically correct model transformation MT based on forward rules. Then, MT is completely information preserving if the corresponding backward model transformation according to Def. 5.1.15 has functional behaviour.*

Proof. Assume that MT is strongly syntactically correct, then we know that $MT(\mathcal{L}_S) \subseteq \mathcal{L}_T$. Assume further that the corresponding backward model transformation has functional behaviour with respect to $\mathcal{L}_T$. Thus, we also have that the backward model transformation has functional behaviour with respect to $MT(\mathcal{L}_S)$ and by Thm. 5.2.37, we directly conclude that MT is completely information preserving. □

Summing up, the analysis techniques and results concerning functional behaviour as presented in this section can be used to ensure unique results, and in addition, they can be used to ensure a complete preservation of information when transforming a model of a source DSL into a model of a considered target DSL. Moreover, the next section on

optimization shows how some of these results are additionally used to improve and ensure efficient executions of model transformations.

5.3 Optimization and Evaluation

Bidirectional model transformations are a key concept for supporting model driven interoperability. The main aim in this context is that models in one domain are maintained separately from models in other domains, while model modifications are reflected and communicated to the interrelated models. For this purpose, the execution of model transformations has to respect certain efficiency constraints. But also in the general case, efficiency plays an important role for ensuring high usability of model transformation techniques.

In order to improve and – in the best case – guarantee efficient executions of model transformations, we present in this section several techniques for the optimization of model transformations based on TGGs, while the fundamental results concerning correctness, completeness, and termination, as well as functional behaviour and information preservation remain valid. We also show how model transformations can be improved by detecting and reducing inadequate translations of certain models based on automated tool support. Finally, we summarize and evaluate the main results of our approach to model transformation according to the listed challenges at the beginning of this chapter.

At first we show in Sec. 5.3.1 how model transformations are analysed in order to detect conflicting pairs of rules. The analysis results support the designer of a model transformation by improving the adequateness and efficiency of the model transformation. Sec. 5.3.2 thereafter explains how the results in Sec. 5.3.1 and, additionally, Sec. 5.2.2 are used for the reduction of backtracking efforts during the execution of a model transformation. By Thm. 5.2.27 in Sec. 5.2.2 we can – in the best case – guarantee executions in polynomial time and space if all significant critical pairs are strictly confluent for the improved rule set. As a benchmark we evaluate our case study concerning execution times and show that the additional overhead caused by the insertion of filter NACs is rather small and does not lead to exponential efforts, while backtracking usually does. In some cases filter NACs suffice to show functional behaviour, and in this case backtracking can be avoided in general (see Thms. 5.2.27 and 5.2.31). For the cases where backtracking is necessary, we additionally provide results for analysing parallel independence (Thm. 5.3.10, Fact 5.3.11), which build the basis for partial order reduction techniques. Based on the results in Ch. 3 we furthermore show how backtracking efforts can be reduced with respect to time and space. Finally, we evaluate our approach to model transformations based on TGGs according to the list of challenges in Sec. 5.1 and show that we provide powerful results for the main part of it.

5.3.1 Detection, Reduction and Elimination of Conflicts

The design of model transformations involves several decisions concerning the interrelationship of the involved DSLs and how the underlying correspondences between the different model elements of two models shall be established by the triple rules. The designer is faced with several important goals at the same time, like adequateness of the model transformation relation with respect to the application domains, efficiency of the execution and - if required - functional behaviour and complete information preservation. For this reason, we now present automated techniques for providing tool support at the design phase. The techniques are based on the results in Sec. 5.2.2 and provide analysis results with respect to the interplay of the triple rules, their possible conflicts and the interpretation, evaluation and resolution of such conflicts. These techniques are additionally used in Sec. 5.3.2 for improving the efficiency of a model transformation.

One important question at the design phase usually is whether a particular triple rule shall create relatively large and complex patterns or, alternatively, rather less complex ones. In most cases the simpler the triple rules are the easier they are to understand. Moreover, compact rules usually reduce the size of the triple graph grammar given by the sum of the type graph and the rule components. However, triple rules sometimes have to be – to a certain extent – larger, because there are interrelationships between DSLs that cannot be achieved by relatively small rules only. In more detail, if the translation of a particular pattern P of a source model cannot be uniquely specified by the compact pattern P itself, because the translation depends on some further context, the rules have to contain some additional context around P as well. In many cases, these situations can be detected by conflict detection, because the variants for a translation are usually specified by different rules and thus, there are at least two rules with a different behaviour on the particular pattern.

In addition to the detection of conflicts the designer needs support for conflict resolution and elimination, respectively. For this purpose we specify a special class of conflicts, called MTR-diverging conflicts, that cause a model transformation to be ambiguous for at least one particular source model G^S, i.e. there are two corresponding target models $G_1^T \ncong G_2^T$ with $(G^S, G_1^T) \in MTR$ and $(G^S, G_2^T) \in MTR$. From the application point of view, model transformations sometimes do not have to ensure functional behaviour, but the possible target models for a given source model have to be semantically equivalent. If this is not the case for the considered target models $G_{1,T}, G_{2,T}$, then the detected conflict needs to be eliminated, because one of the target models is not intended to be a valid result. By Thm. 5.3.7 we can ensure that the technique of conservative conflict resolution ensures that the derived new model transformation relation is contained in the original one. This ensures that no new pairs are inserted by the rule modifications and thus, the amount of invalid translations from the semantic point of view is reduced.

Conflicts of transformation steps are caused by the non-deterministic choice concerning the current rule and match for the next transformation step during the execution of a transformation. This general flexible concept for graph transformation systems has the advantage that the designer of a model transformation does not have to specify a detailed and possibly complex control structure, which completely determines the order of the steps for each possible input model. Moreover, the model transformation can be build up iteratively rule by rule, i.e. feature by feature, and each transformation rule can be maintained more or less separately when the involved DSLs evolve or other context constraints change.

But the non-determinism may cause some side effects of some rules with respect to other rules, because the application of one rule may exclude the execution of another rule from the possible continuations of the transformation sequence. In the general case this may lead to non-functional behaviour. Furthermore, even if functional behaviour is not required, there may be parts of models that should always be translated by one specified transformation rule. Vice versa, there may be specific parts of models that should never be translated by one particular rule.

We now show by the following example how the conflict analysis techniques for model transformation rules are used to provide visualized results to the designer, who may become aware of possibly unintended behaviour of the concrete model transformation at certain models. The analysis is based on the presented techniques in Sec. 5.2.2 for analysing functional behaviour based on critical pairs [EEPT06, Lam09]. The visualized conflicts are presented in minimal context, and the designer has the choice whether to keep the model transformation unchanged or to modify some transformation rules by e.g., the insertion of derived NACs that exclude the detected conflict.

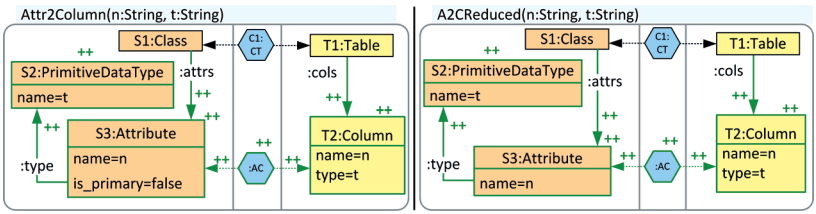


Figure 5.21: Original TGG-rule "$Attr2Column$" and modified rule "$A2CReduced$"

Example 5.3.1 (Conflict detection)**.** *During the design of the model transformation CD2RDBM the modeller may inspect the rule "$Attr2Column$" (left of Fig. 5.21) and observe that value of the attribute "$is\_primary$" of node "S3" is not used for the corresponding part in the data base. He concludes that the attribute could be handled separately by another rule, which would add the Boolean value "$is\_primary = false$" in the class diagram and would keep the data base unchanged. The resulting reduced rule "$A2CReduced$"*

is shown on the right of Fig. 5.21. In order to check the effect of this change for the behaviour of the model transformation we can now use the critical pair analysis engine of AGG [AGG10] for the derived forward translation rules as described in Sec. 5.2.2. AGG can purely generate the conflicts of rules that contain the new rule. There are three critical pairs concerning attributes (attribute-change) and three concerning NACs (produce-forbid). The first three only differ on how many elements are commonly matched, and the remaining three can be neglected because they contain overlapping graphs, which violate the language constraints of the source language. Thus, we show one representative critical pair in Fig. 5.22. In Ex. 5.3.3 we explain how the designer can interpret the situation and how he can solve the problematic situation.

Remark 5.3.2 (Creation of nodes with attributes)**.** *Note that the reduced rule "A2CReduced" in Ex. 5.3.1 is well-defined, although one attribute of "S3" is not set. This is possible by the underlying concept of E-graphs (See Sec. 3.1 and [EEPT06]) for attributed graphs, which allows for matching and creation of nodes where not all attributes are set. However, rules can also be restricted to create only nodes where all attributes are specified. In this case we can imagine another example with a more complex rule, where some additional context is not needed for the translation directly, but for the restriction of matches. In the simplest case consider for example, that the attribute "is_primary" is equivalently specified within the abstract syntax by an external node. This means that the type graph would contain the additional node type "Is_primary", and the rule "A2CReduced" would respect the requirement that all attributes of created nodes are set.*

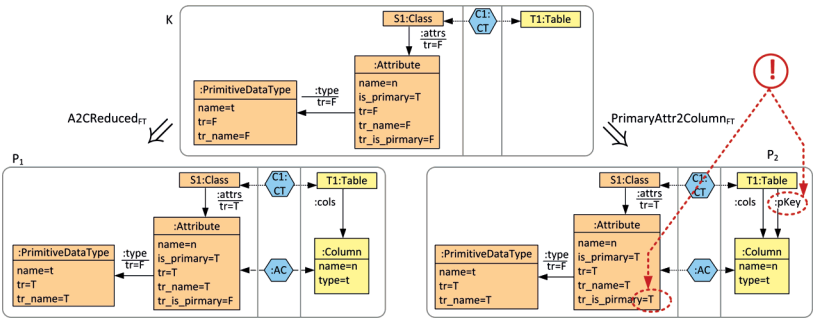


Figure 5.22: Critical pair for *TGG*-rules modified rule "*A2CReduced*" and "*PrimaryAttr2Column*"

Example 5.3.3 (Conflict Resolution)**.** *The critical pair in Fig. 5.22 shows a conflict between the modified rule "A2CReduced" and "PrimaryAttr2Column". Obviously, the*

graph P_1 shows a pattern that is problematic, because the node of type "$Attribute$" with attribute value "$is\_primary = true$" is already translated, but the corresponding column is not marked as primary attribute. The designer has the following options.

1. *The overlapping graph K can be used as additional NAC of rule "$A2CReduced$" to avoid an application in this context. AGG offers a direct extension of the rule with K as NAC. The conflict analysis has to be executed again to detect new conflicts that may appear. The resulting behaviour of "$A2CReduced$" becomes more or less equivalent to the original rule "$Attr2Column$", and the model transformation relation is the same as for the original system. This conflict resolution gives rise to the general notion of conservative conflict resolution, which is defined in Def. 5.3.6 later on in this section.*

2. *Rule "$PrimaryAttr2Column$" can by reduced now to a rule that only translates the attribute "$is\_primary = true$" of a node of type "$Attribute$". Furthermore, there would be the need for an additional rule that create the attribute value "$is\_primary = false$" without creating an edge of type "$pkey$" in the target component.*

3. *Since the designer is aware of the conflict, which shows an unintended behaviour of the model transformation, he can decide to keep the original rule and delete the modified rule "$A2CReduced$".*

Which decision the designer takes depends on the further possible extensions he has in mind. All three possibilities will lead to the same model transformation relation as in the original system.

As shown by Examples 5.3.1 and 5.3.3, the design decisions within the modelling process can be supported by automated techniques for conflict detection. Most importantly, the automated techniques can detect unintended side effects of transformation rules and concretely visualize the conflicts in minimal contexts. We now specify a class of conflicts between TGG-rules that are similar to the conflicts of the example. Analogously to the analysis of functional behaviour in Sec. 5.2.2, we define a special class of critical pairs that show a so-called MTR*-divergence conflict*. If a critical pair shows an MTR*-divergence conflict*, we know that there is a pattern of a source model that can be translated into two different ways leading to different resulting target models. In this situation, the designer has to judge whether both possibilities are intended, because they are e.g. semantically equivalent in the target domain, or whether one of them is not intended and should be eliminated. An elimination can be performed directly by specifying the overlapping graph K as NAC for the rule of the unintended step.

Definition 5.3.4 (MTR-divergence Conflict)**.** *Let MTR be the model transformation relation of a model transformation based on forward translation rules. A critical pair $P_1 \xleftarrow{tr_{1,FN}} K \xrightarrow{tr_{2,FN}} P_2$ shows an MTR-divergence Conflict, if it can be embedded in a pair $G'_{i+1} \xleftarrow{tr_{1,FN}} G'_i \xrightarrow{tr_{2,FN}} G''_{i+1}$, where $(G^S, G'_0 \xRightarrow{tr^{i,*}_{FN}} G'_i \xRightarrow{tr_{1,FN}} G'_{i+1} \xRightarrow{tr^{j,*}_{FN}} G'_n, G^T)$ and $(G^S, G'_0 \xRightarrow{tr^{i,*}_{FN}} G'_i \xRightarrow{tr_{2,FN}} G''_{i+1} \xRightarrow{tr^{k,*}_{FN}} G''_m, \overline{G}^T)$ are model transformation sequences with $G^T \ncong \overline{G}^T$.*

Remark 5.3.5 (Analysis of MTR-divergence Conflicts)**.** *In Sec. 5.2.2 we presented how AGG [AGG10] is used for the analysis of critical pairs based on the flattening construction. Besides the full generation of all critical pairs, AGG can optionally generate the critical pairs for some particular rule combinations, thus we can restrict the analysis to the rule that is currently developed by the designer. Given a critical pair $(P_1 \Leftarrow K \Rightarrow P_2)$, the question now is whether it shows an MTR-divergence conflict or not.*

A critical pair does not show an MTR-divergence conflict if P_1, P_2 or both are misleading graphs according to Def. 5.2.17. By Fact 5.2.21 we have presented a static generation technique for filter NACs which can be used as well to generate a set of misleading graphs in order to check whether P_1 or P_2 are included showing that the critical pair does not show an MTR-divergence conflict.

Vice versa, given a generated critical pair, we can start a search for corresponding model transformation sequences by taking the source component K^S of K without translation attributes and extending it to possible source models G^S on which we can execute the model transformation and store the found sequences. If the procedure finds two diverging sequences where the critical pair is embedded the designer of the model transformation can be notified that the critical pair shows an MTR-divergence conflict.

Once a particular conflict is detected it can be resolved in a conservative way as defined below, where one rule is extended by a new NAC in order to forbid the application of this rule in situations where the conflict may occur independent of additional context. The overlapping graph is used to specify this NAC.

Definition 5.3.6 (Conservative Conflict Resolution)**.** *Let $(P_1 \xleftarrow{tr_{1,FN},m_1} K \xrightarrow{tr_{2,FN},m_2} P_2)$ be a critical pair of the set TR_{FN} containing forward translation rules with filter NACs. A new set TR'_{FN} is derived from TR_{FN} by conservative conflict resolution, if $TR'_{FN} = (TR_{FN} \setminus \{tr_{1,FN}\}) \cup \{tr'_{1,FN}\}$, where $tr'_{1,FN}$ is given by $tr_{1,FN}$ extended with a new NAC $(n : L_{1,FN} \to N)$ with $n = m_1$ and $N = K$.*

By Thm. 5.3.7 we show that the new induced model transformation relation conforms to the original one in the way that only some pairs may be deleted, but no new pair is inserted. Since the additional NAC restricts the rule application for an arbitrarily bigger context, there may be several pairs of MTR that are deleted, and if the NAC cannot be

found in an valid intermediate model during the execution of the model transformation, then the model transformation relation remains unchanged after the insertion of the NAC.

Theorem 5.3.7 (Conservative Conflict Resolution)**.** *Given a model transformation based on forward translation rules with filter NACs TR_{FN} with model transformation relation MTR. Let TR'_{FN} be derived from TR_{FN} by stepwise conservative conflict resolution, then the corresponding new model transformation relation MTR' for TR'_{FN} is included in the original model transformation relation MTR:*

$$\boxed{MTR' \subseteq MTR}$$

Proof. Let $(G^S, G^T) \in MTR'$. This implies that there is a model transformation sequence $(G^S, G'_0 \overset{tr'^{*}_{FT}}{\Longrightarrow} G'_n, G^T)$. MTR' is based on TR' and TR' contains the same rules as TR, but each rule may contain some additional NACs. Therefore, the TGT-sequence $(G'_0 \overset{tr'^{*}_{FT}}{\Longrightarrow} G'_n) = (G'_0 \overset{tr'_{1,FT},m_{1,FT}}{\Longrightarrow} G'_1 \overset{tr'_{1,FT},m_{1,FT}}{\Longrightarrow} \dots G'_{n-1} \overset{tr'_{n,FT},m_{1,FT}}{\Longrightarrow} G'_n)$ based on TR' implies the sequence $(G'_0 \overset{tr^{*}_{FT}}{\Longrightarrow} G'_n) = (G'_0 \overset{tr_{1,FT},m_{1,FT}}{\Longrightarrow} G'_1 \overset{tr_{1,FT},m_{1,FT}}{\Longrightarrow} \dots G'_{n-1} \overset{tr_{n,FT},m_{1,FT}}{\Longrightarrow} G'_n)$ based on TR. Thus, $(G^S, G^T) \in MTR$. □

As described in Rem. 5.3.5 above, we presented some techniques for the analysis of conflicts and in particular, an analysis to check whether certain detected conflicts are MTR-divergence conflicts. By Thm. 5.3.7 we can ensure that a conservative resolution of conflicts of the critical pairs generated by the tool AGG may only reduce the model transformation relation by eliminating some pairs of source and target models. This means that no new pairs can be introduced, which is important for the design of model transformations. This way, the designer is enabled to first create model transformation rules that are perhaps too flexible, and he can restrict their application thereafter without any risk that he may insert new possible translation variants for some source models.

5.3.2 Reduction and Elimination of Backtracking

While many functional properties, such as syntactical correctness and completeness of model transformations based on TGGs can be ensured in general as presented in Sec. 5.2.1, this is not the case for the efficiency of its execution caused by the non-deterministic nature of graph transformation systems in general. However, efficiency plays a big role when applying model transformations in running modelling environments with a variety of DSLs.

In order to improve the efficiency of model transformations based on TGGs we mainly have to reduce the costs of backtracking. For this purpose we present in this section different techniques which help to reduce the amount of the cases for backtracking by cutting off backtracking paths and filtering out equivalent ones. Moreover, we show how the memory consumption can be kept relatively low even if the models itself are big.

Here we can again reuse results from previous sections. First of all, the reduction of backtracking cases is based on the detection and elimination of conflicts in Sec. 5.3.1 and, moreover, based on the results on functional behaviour in Sec. 5.2.2 we can avoid backtracking completely if some sufficient conditions hold. Based on the results on behaviour analysis of general graph transformation systems in Ch. 3 we can, furthermore, filter out equivalent transformation paths and reduce the memory consumption by constructing the corresponding subobject transformation system (STS) allowing us to omit the storage of intermediate models.

Filter NACs introduced in Sec. 5.2.2 on the one hand support the analysis of functional behaviour, and on the other hand, also improve the efficiency of the execution. By definition, the occurrence of a filter NAC at an intermediate model means that the application of the owning rule would lead to a model that cannot be translated completely, i.e. the execution of the model transformation would perform backtracking at a later step. This way, a filter NAC *cuts off possible backtracking paths* of the model transformation. As presented in Fact 5.2.21 some filter NACs can be generated automatically, and using Fact 5.2.24, a larger set of them can be obtained based on the generation of critical pairs.

As shown by Thm. 5.2.27, we can completely avoid backtracking if all critical pairs of the system of forward translation rules, possibly enriched by additional filter NACs, are strictly confluent. Furthermore, we have shown by Thm. 5.2.31 that backtracking can be avoided as well, if the constructed filter NACs lead to a set of rules TR_{FN} that has no significant critical pair.

Model Size	Model Transformation Sequences of CD2RDBM				
	without Filter NACs		with Filter NACs		
	Time[1)]	Success Rate	Time[1)]	Overhead	Success Rate
[Elements[2)]]	[ms]	[%]	[ms]	[%]	[%]
11	144	42,86	158	10,14	100,00
25	303	16,84	335	10,80	100,00
53	673	3,94	743	10,40	100,00
109	1.481	0,17	1.585	6,98	100,00

1) Average time of 100 successful model transformation sequences
2) Nodes and Edges

Table 5.2: Benchmark, Tool: AGG [AGG10]

Example 5.3.8 (Benchmark: Execution of Model Transformations)**.** *Table 5.2 shows execution times using the transformation engine AGG [AGG10]. The additional overhead caused by filter NACs is fairly small and lies in the area of 10% for the examples in the benchmark, which is based on the average execution times for 100 executions concerning models with 11, 25, 53 and 109 elements (nodes and edges), respectively. The first model with 11 elements is the presented class diagram in the source component of Fig. 5.10. We*

explicitly do not compare the execution times of the system with filter NACs with one particular system with backtracking, because these times can vary heavily depending on the used techniques for partial order reduction and the chosen examples. Instead we present the computed success rates for the system without NACs, which show that backtracking will cause a substantial overhead in any case. Thus, the listed times concern successful execution paths only, i.e. those executions that lead to a completely translated model. The success rate for transformations without filter NACs decreases fast when considering larger models, and in particular, for the model with 109 elements the success rate is below 0.2%. This means that the error rate is above 99.8%, while the success rate for the system with filter NACs is always 100%.

The system without filter NACs requires backtracking in order to ensure completeness, because there are transformation paths which do not lead to completely translated models. The times for the unsuccessful executions that end at a partially translated model are not measured, i.e. the additional overhead of backtracking is not contained in the numbers. This overhead is caused on the one hand by the execution of unsuccessful transformation sequences and on the other hand by the additional efforts for storing the transformation details in order to restart the transformation process at intermediate steps. In the worst case, backtracking leads to the complete construction of all possible transformation sequences for a given source model, i.e. the construction of the full derivation tree, because the successful path may be constructed as the last one or the source model cannot be translated at all, because it does not belong to VL_S. Even if partial order reduction techniques are applied to remove equivalent paths, the worst case still leads in general to a derivation tree implying that the complexity of backtracking is still exponential with respect to the length of one transformation sequence for the given input model. In our case study misleading graphs appear already at the beginning of many transformation sequences. This implies that backtracking will cause in the worst case the construction of wide derivation trees, which is costly.

Backtracking for general model transformation systems is reduced by filter NACs and avoided completely in the case that no "significant critical pair" remains present (see Thm. 5.2.31), which we have shown to be fulfilled for our example. The additional overhead of about 10% for filter NACs is in most cases much smaller than the efforts for backtracking.

Moreover, in order to perform model transformations using highly optimized transformation machines for plain graph transformation, such as Fujaba and GrGen.Net [TBB+08], we have presented how the transformation rules and models can be equivalently represented by plain graphs and rules. First of all, triple graphs and morphisms are flattened according to the construction presented in Sec. 5.2.2 and in [EEH08c, HEOG10]. Furthermore, we presented how forward rules with NACs are extended to forward translation rules with NACs, such that the control condition "source consistency" [EEE+07] and the gluing

condition (Fact 5.1.37) are ensured automatically for complete sequences, i.e. they do not need to be checked during the transformation.

In order to perform partial order reduction when backtracking cannot be completely avoided, we now present several results concerning parallel independence.

For model transformations based on forward rules we can analyse the execution paths during the on-the-fly construction presented in Sec. 5.1.1. Two partially match consistent sequences which differ only in the last rule application are parallel independent if the last rule applications are parallel independent both for the source and forward sequence, and, in addition, if the embeddings into the given graph G_0 are compatible.

Definition 5.3.9 (Parallel Independence of Partially Match Consistent Extensions)**.** *Two partially match consistent sequences*
$\varnothing = G_{00} \overset{tr_S^*}{\Longrightarrow} G_{n0} \overset{tr_{1,S}}{\Longrightarrow} G_{n+1,0} \overset{g_{n+1}}{\hookrightarrow} G_0 \overset{tr_F^*}{\Longrightarrow} G_n \overset{tr_{1,F}}{\Longrightarrow} G_{n+1}$ *and*
$\varnothing = G_{00} \overset{tr_S^*}{\Longrightarrow} G_{n0} \overset{tr_{2,S}}{\Longrightarrow} G'_{n+1,0} \overset{g'_{n+1}}{\hookrightarrow} G_0 \overset{tr_F^*}{\Longrightarrow} G_n \overset{tr_{2,F}}{\Longrightarrow} G'_{n+1}$
are parallel independent *if* $G_{n0} \overset{tr_{1,S}}{\Longrightarrow} G_{n+1,0}$ *and* $G_{n0} \overset{tr_{2,S}}{\Longrightarrow} G'_{n+1,0}$ *as well as* $G_n \overset{tr_{1,F}}{\Longrightarrow} G_{n+1}$ *and* $G_n \overset{tr_{2,F}}{\Longrightarrow} G'_{n+1}$ *are parallel independent leading to the diagram* (1_S) *and* (1_F)*, and diagram* (2) *is a pullback.*

$$\begin{array}{ccc} G_{n0} & \overset{tr_{1,S}}{\Longrightarrow} & G_{n+1,0} \\ {\scriptstyle tr_{2,S}}\Downarrow & (1_S) & \Downarrow{\scriptstyle tr_{2,S}} \\ G'_{n+1,0} & \overset{tr_{1,S}}{\Longrightarrow} & G_{n+2,0} \end{array} \qquad \begin{array}{ccc} G_n & \overset{tr_{1,F}}{\Longrightarrow} & G_{n+1} \\ {\scriptstyle tr_{2,F}}\Downarrow & (1_F) & \Downarrow{\scriptstyle tr_{2,F}} \\ G'_{n+1} & \overset{tr_{1,F}}{\Longrightarrow} & G_{n+2} \end{array} \qquad \begin{array}{ccc} G_{n0} & \overset{t_{1,S}}{\hookrightarrow} & G_{n+1,0} \\ {\scriptstyle t_{2,S}}\downarrow & (2) & \downarrow{\scriptstyle g_{n+1}} \\ G'_{n+1,0} & \overset{g'_{n+1}}{\hookrightarrow} & G_0 \end{array}$$

In the case of parallel independence of the extensions, both extensions can be extended both in the source and forward sequences leading to two longer partially match consistent sequences which are switch-equivalent.

Theorem 5.3.10 (Partial Match Consistency with Parallel Independence)**.** *If* $\varnothing = G_{00} \overset{tr_S^*}{\Longrightarrow} G_{n0} \overset{tr_{1,S}}{\Longrightarrow} G_{n+1,0} \overset{g_{n+1}}{\hookrightarrow} G_0 \overset{tr_F^*}{\Longrightarrow} G_n \overset{tr_{1,F}}{\Longrightarrow} G_{n+1}$ *and* $\varnothing = G_{00} \overset{tr_S^*}{\Longrightarrow} G_{n0} \overset{tr_{2,S}}{\Longrightarrow} G'_{n+1,0} \overset{g'_{n+1}}{\hookrightarrow} G_0 \overset{tr_F^*}{\Longrightarrow} G_n \overset{tr_{2,F}}{\Longrightarrow} G'_{n+1}$ *are* parallel independent *then the following upper and lower sequences are partially match consistent and called* switch equivalent*.*

$$\begin{array}{ccccccccc} & & & & G_{n+1,0} & & & & \\ & & & {\scriptstyle tr_{1,S}}\nearrow & & \searrow{\scriptstyle tr_{2,S}} & & & \\ \varnothing = G_{00} & \overset{tr_S^*}{\Longrightarrow} & G_{n0} & & & & G_{n+2,0} & \hookrightarrow & G_0 \\ & & & {\scriptstyle tr_{2,S}}\searrow & & \nearrow{\scriptstyle tr_{1,S}} & & & \\ & & & & G'_{n+1,0} & & & & \end{array} \overset{tr_F^*}{\Longrightarrow} \begin{array}{ccc} & G_{n+1} & \\ {\scriptstyle tr_{1,F}}\nearrow & & \searrow{\scriptstyle tr_{2,F}} \\ G_n & & G_{n+2} \\ {\scriptstyle tr_{2,F}}\searrow & & \nearrow{\scriptstyle tr_{1,F}} \\ & G'_{n+1} & \end{array}$$

Proof. The proof of Thm. 5.3.10 above is given by the proof for Thm. 5 in [EEHP09a]. □

While we construct the source and forward sequences separately for model transformations based on forward rules using the on-the-fly construction, they are constructed in an

integrated way for model transformations based on forward translation rules. Therefore, we can reduce the analysis of parallel independence to the common notion of parallel independence for typed attributed graph transformation systems as shown by Fact. 5.3.11 below.

Fact 5.3.11 (Parallel Independence of Forward Translation Steps)**.** *Given a TGT-sequence* $G'_0 \overset{tr^*_{FT}}{\Longrightarrow} G'_n$ *and let* $G'_n \overset{tr_{FT,1},m'_{1,FT}}{\Longrightarrow} H'_1$ *and* $G'_n \overset{tr_{FT,2},m'_{2,FT}}{\Longrightarrow} H'_2$ *be two parallel independent transformation steps according to Def. 3.2.4 in Sec. 3.2. Then, there are sequentially independent transformation steps*

$H'_1 \overset{tr_{FT,2},m'_{2a,FT}}{\Longrightarrow} H'$ *and* $G'_2 \overset{tr_{FT,1},m'_{1a,FT}}{\Longrightarrow} H'$.

Proof. This result follows directly by the local Church-Rosser Thmeorem for typed attributed graph transformation systems with NACs as shown by Thm. 3.4.3 in [Lam09] for weak adhesive HLR systems, and we know by Thm. 11.11 in [EEPT06] that typed attributed graphs form a weak adhesive HLR category. Note that the distinguished class $\mathcal{Q}$ in [Lam09] coincides with $\mathcal{M}$ for the notion of typed attributed graph transformation systems in this thesis, which is also the usual case. Note further that the notion of $\mathcal{M}$-adhesive categories is a generalisation of weak adhesive HLR categories. □

During the execution of a model transformation based on forward translation rules we can apply the analysis techniques of Ch. 3 and construct the subobject transformation system (STS) of the current transformation sequence. An STS $\mathcal{S} = (T, P, \pi)$ can be used to store all relevant information of the transformation sequence in a compact way such that the possibly quite complex intermediate models and relating morphisms do not have to be stored separately. The sequence $s = \langle r_1; \ldots; r_n \rangle$ of rule names in P specifies the executed steps of the transformation sequence and using the STS backtracking is possible to all intermediate steps. Furthermore, the STS is constructed iteratively, and therefore, it can be updated iteratively for the current considered transformation sequence within the derivation tree during the backtracking phase.

A further improvement of backtracking can be achieved by generating the dependency net of the constructed STS, which can be used to generate all permutation-equivalent paths. Depending on the strategy for partial order reduction the derived information can be used to reduce the amount of explicit transformation steps that have to be executed.

Summing up, the presented results on parallel independence (Thm. 5.3.10 Fact 5.3.11) provide a formal basis for partial order reduction techniques. The construction of the STS during the backtracking phase enables a further reduction of the required space and therefore, it can reduce the execution time by reducing the efforts for write and read operations that have to be performed. As main contribution we have shown by the benchmark in Ex. 5.3.8 that the generation and specification of filter NACs generally reduces backtracking, and in some cases, filter NACs ensure strong functional behaviour of the model trans-

formation such that backtracking is avoided completely. Using the flattening construction as explained in Sec. 5.2.2 we can use the best available tools for executing graph transformations including the available partial order reduction techniques, while still ensuring correctness and completeness.

5.3.3 Evaluation of the Approach

By comparing the listed challenges at the beginning of this chapter concerning properties of the functional and non-functional dimensions of model transformations, we can state that the presented approach of model transformations based on TGGs provides important and powerful results that encompass the major part of the specified challenges. In the following we will summarize the main achievements in this direction.

1. *Syntactical Correctness:* Model transformations based on forward rules and equivalently those based on forward translation rules (see Sec. 5.1) are syntactically correct with respect to the language $VL = \{G \mid \emptyset \Rightarrow^* G \text{ in } TGG\}$ containing the integrated models generated by the triple rules (Thms. 5.2.1 and 5.1.22 and Cor. 5.2.6).

2. *Semantic Correctness:* Techniques for analysing the semantic correctness of a model transformation are not in the focus of this thesis. First results in this direction are presented in [HHK10], where we prove the semantic correctness of a rather simple model transformation, and we show how this approach has the potential to be generalized to other model transformations as well.

3. *Completeness:* Model transformations based on forward rules and equivalently those based on forward translation rules (see Secs. 5.1) are complete with respect to the language VL_S of source models and VL_T of target models (Thms. 5.2.1, 5.2.4 and 5.1.22 and Cor. 5.2.6).

4. *Functional Behaviour:* The presented techniques for analysing functional behaviour in Sec. 5.2.2 are general, powerful and distinguish between functional behaviour (Thm. 5.2.27) in general and strong functional behaviour (Thm. 5.2.31), where also the model transformation sequences are unique up to switch-equivalence. The techniques are based on the generation of translation attributes, and we provide tool support by the critical pair analysis engine of AGG [AGG10].

The following list describes the achievements concerning the *non-functional dimension* about usability and applicability aspects of model transformations.

1. *Efficiency:* The on-the-fly-construction in Sec. 5.1.1 provides an efficient execution of source consistent model transformations for which termination is ensured if the

source rules are creating. As a second optimization, suitable conditions for parallel independence were defined in [EEHP09b] in order to perform partial order reductions. The efficiency is further improved in Sec. 5.3.2 by providing a sufficient condition to avoid backtracking completely. The presented benchmark shows that the additional filter-NACs produce an overhead of approx. 10 %, such that the execution in the case without backtracking stays within the required bound of polynomial space and time.

2. *Intuitive Specification:* TGGs provide a pattern based specification that allows the designer to intuitively define the model transformation rules based on example fragments of integrated models. In many cases, the concrete syntax of visual languages can be used for the specification of triple rules as long as all effects of a transformation rule are visible within the concrete syntax. In any case, triple rules in abstract syntax can be visualized using the concrete syntax in order to provide a better intuition to the designer.

3. *Maintainability:* As described in Sec. 5.3.1, model transformation rules can be inserted and modified in a sophisticated way, where side effects of local changes can be analysed automatically, and detected conflicts can be resolved with tool support (see Thm. 5.3.7).

4. *Expressiveness:* NACs are commonly used in order to define expressive model transformations, and therfore, we integrated NACs in our approach (see Sec. 5.1). Meanwhile, model transformations based on TGGs were further extended to the more general nested application conditions [GEH11, HP09], which provide the expressive power of first order logic on graphs. Since the special control condition source consistency that ensures syntactical correctness and completeness is handled automatically in our approach, we do not need to specify additional control structures for this purpose.

5. *Bidirectional model transformations:* Model transformations based on forward rules and equivalently, those based on forward translation rules (see Secs. 5.1), are bidirectional. Their execution does not change the given models, which especially important for model based interoperability and for distributed coevolution of models. Furthermore, our approach to model transformation ensures information preservation, and we can ensure complete information preservation (see Thms. 5.2.34 and 5.2.37) for the case that the corresponding backward transformation is terminating and has functional behaviour.

Summing up, the presented approach to model transformations based on triple graph grammars provides an intuitive, expressive, formally well-founded and efficient framework for bidirectional model transformations including powerful results for analysis and

optimization. According to the listed achievements above there are several important advantages in comparison to other existing approaches, like [SK08, KS06, KW07, GW09, GH09], which are mainly software engineering focused, and therefore, do not offer similar formal results. However, these approaches are very similar and stimulated the development of some constructions in this thesis. This means that there is a basis to transfer the presented results to the related approaches with some modification efforts. In Ch. 7 we present the details of a implemented prototypical reference model transformation engine, which can be used to test and adapt available tools that have been optimized already with respect to efficiency.

Chapter 6

Conformance Analysis of Enterprise Process and Service Models

Besides the modelling of relevant business processes as presented in Ch. 4 there are several further dimensions in enterprise modelling as already described in Ch. 2. All dimensions of enterprise modelling can overlap and influence each other. In particular, business process models have to conform to the specified guidelines and structure of the enterprise. For this reason, we present in this chapter how conformance of business process models to a given business service structure model can be analysed based on automated formal techniques and tool support using the formal results for model transformation in Ch. 5.

At first, Sec. 6.1 introduces business service structure models (BSS models). They can be seen as integrated models which are obtained from the following types of models. Structure models are used to integrate the specified hierarchy or matrix of the actors (persons and software applications) of the enterprise. Moreover, we use models which specify the communication channels in order to restrict communication to these channels. Finally, BSS models integrate the specification of access rights, i.e. models for the definition of permissions concerning read and write access rights of actors to resources.

Section 6.2 presents the triple graph grammar which relates WDEPC business process models as given in Ch. 4 and business service structure models. This way, we also derive a bidirectional model transformation between both model types by generating the forward and backward model transformations from the specified TGG. Based on the results for TGGs in Ch. 5, we formalize the notion of conformance in Sec. 6.3 and show by Thm. 6.3.3 how conformance is analysed with formal and automated techniques. Moreover, we show that the TGG of Sec. 6.2 satisfies the conditions of Thm. 6.3.3, which allows us to show that the business process model of the case study conforms to the business service structure model. Furthermore, by Def. 6.3.8 we can directly apply Thm. 6.3.3 for showing that the business process model with additional continuity snippets conforms to the business service structure model. By Cor. 6.3.4, we additionally show how service structure models

are generated out of given business process models, such that conformance of the business process models to the generated service structure models is guaranteed. Therefore, the generated models can be used for further refinement in the modelling process. Finally, Sec. 6.4 presents some optimization techniques and strategies for conformance analysis and shows by Thm. 6.4.1 how conformance checks can be ensured to be efficient based on the results in Ch. 5.

6.1 Business Service Modelling

Business service structure models (BSS models) are introduced in this section to provide specifications concerning, on the one hand, the available actors and resources of an enterprise and, on the other hand, concerning the provided communication channels as well as the permitted access to resources. In the case study of this chapter we consider one combined service structure model which can be obtained by integrating the different available visual and textual models and specifications in this context, respectively.

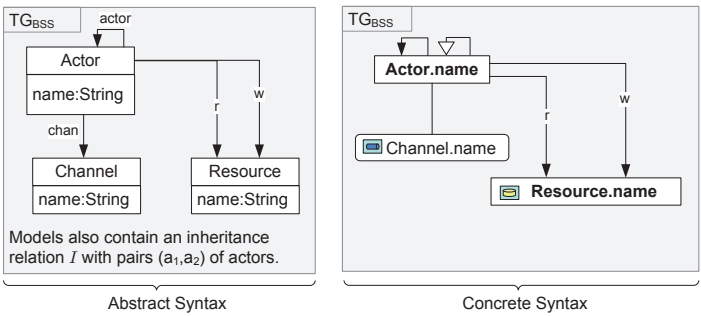


Figure 6.1: Type graph for business service structure models (BSS models)

According to the meta model in Fig. 6.1, given by the type graph TG_{BSS}, BSS models consist of actors, channels for communication between actors and resources to which actors may have access. Furthermore, BSS models specify an inheritance relation I of actor roles, which is visualized in concrete syntax by lines with white arrowheads. This way, BSS models consist of the following components, which are explicitly emphasized in Fig. 6.2 showing the example BSS model of our case study.

The first component (top most part of the figure) defines the inheritance relation between the different kinds of actors. Since the actors are also involved in the further components the actors are depicted again at the relevant places, but they do only occur once in the

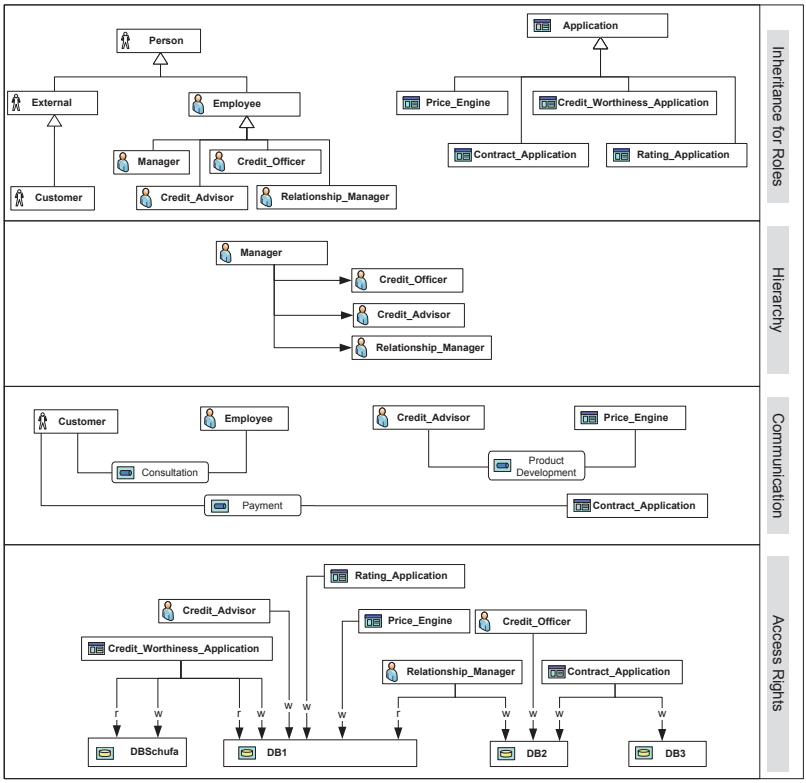


Figure 6.2: Visual business service structure model (BSS model)

abstract syntax as it is common praxis in visual modelling languages, like also in UML models.

Organizational structure models form the second component of BSS models, which are given by common organizational charts [Cha69, VIS09] specifying the structure of the actors. While the example of our case study shows a hierarchical structure the more general case also concerns so-called matrix structures (organigraphs [Min79]), i.e. structures that form a graph with at most one edge between two nodes, but not necessarily a tree.

The third component of BSS models concerns the communication between actors, where communication channels are specified that shall be provided by additional IT support. This way, the possible information flow between different groups and departments of an enter-

prise are specified explicitly and unspecified data flow can be detected by the system and either blocked or only logged for a possible later evaluation. By default, each actor has its own private channel and only the additional channels for communication with other actors are shown in concrete syntax. Note that we present in Sec. 6.2 how the required channels can be generated out of the available business process models and refined thereafter, such that the modelling efforts can be kept low.

Finally, the last component, shown in the lower part of the figure, supports the specification of access rights, i.e. the definition of permissions concerning read and write access rights of actors to resources. The access rights in our case study are not further differentiated concerning, e.g., location and legislative dependencies as in [SE09], but the BSSS models can be extended accordingly as well.

Example 6.1.1 (Visual Business Service Structure Model)**.** *The BSS model in Fig. 6.2 specifies the simplified organizational structure of a team in a financial institution. Head of the team is the manager and he is responsible for the credit officer, the credit advisor and the relationship manager. The model specifies some communication channels, e.g. the customer may have contact to any employee via the channel "consultation". This way, the inheritance relation reduces the modelling efforts and increases the conciseness. The lower part shows the specified access rights, where for instance a relationship manager has read access to data base "DB1", write access to "DB2", but he does not have access to "DBSchufa" or "DB3".*

The different aspects of a BSS model can also be kept in separate models. In this case conformance of business process models can be checked separately for each of the models concerning the chosen aspect only.

6.2 Model Transformation: Business Process to Business Service Models

Based on the presented language of business service structure models in Sec. 6.1 we show in this section how (type restricted) WDEPC models as presented in Sec. 4.1.1 and BSS models are related via a triple graph grammar, from which we derive a bidirectional model transformation $Process2Service$ between both languages in order to analyse conformance in Sec. 6.3. Since there are some elements in an WDEPC model which are independent of the related BSS models and vice versa, we use a generalized notion of conformance, where a given WDEPC model is first restricted to form a model of the source language VL_S of the triple graph grammar and the target language VL_T does only contain a subset of all BSS models.

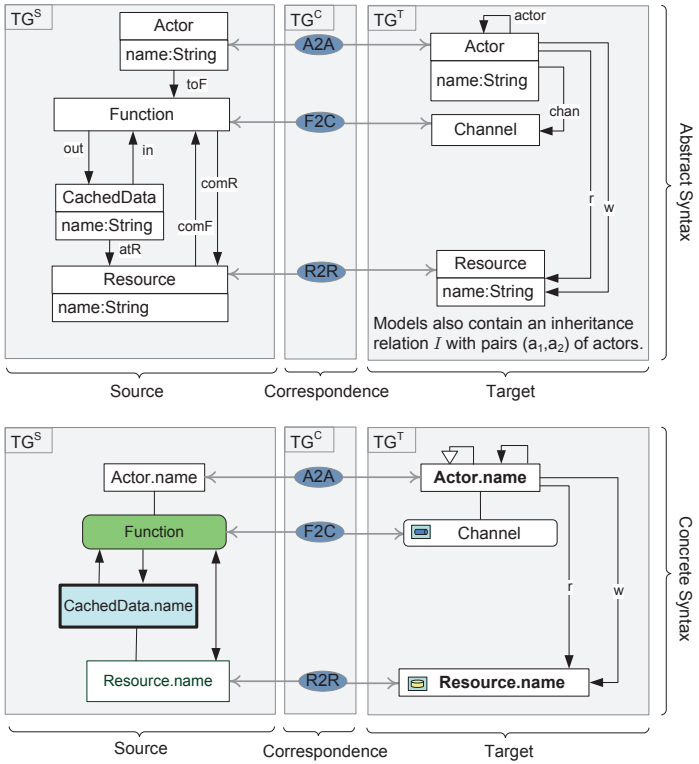


Figure 6.3: Triple Type graph for the model transformation *Process2Service*

According to Ch. 5 a triple graph grammar $TGG = (TG, SG, TR)$ consists of a type graph TG, a start graph SG and a set of triple rules TR. The type graph of our case study for conformance analysis is shown in Fig. 6.3 and the start graph is the empty graph as usual for TGGs. First of all, actors and resources occur on both sides, i.e. the source and target component, and they are related with each other. Based on the interaction of actors at certain functions the communication channels are derived and related to the functions at which they are used. Finally, the edges for read and write access rights ("r" and "w") specify the actors who have permission to send or retrieve data from certain resources as specified in WDEPC models.

In order to consistently relate WDEPC and BSS models we slightly extend the WDEPC models in the way that the labels occurring in a WDEPC model are made explicit by at-

tributes “name:String” in the abstract syntax. Note further that some elements of the meta model TG_{Meta} for WDEPC models in Ch. 4 are not relevant for the conformance check in Sec. 6.3 and therefore, they do not occur in the triple type graph. In order to check conformance the given WDEPC models will be restricted first to the relevant types of elements according to [EEEP09] as described in Def. 6.3.6.

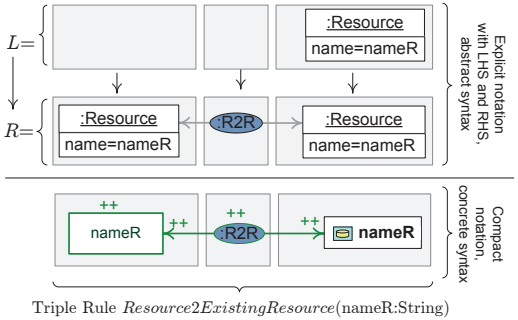


Figure 6.4: Triple rule in explicit and compact notation

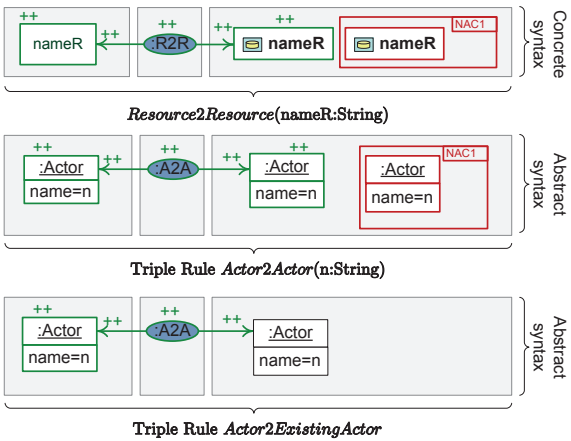


Figure 6.5: Further triple rules for actors and resources

Example 6.2.1 (Triple Rules for the Model Transformation *Process2Service* (First part))**.** *The triple rules of the TGG of our case study are shown in Figs. 6.4 to 6.6 and they are*

typed over the triple type graph in Fig. 6.3. Rule "Resource2ExistingResource" in Fig. 6.4 specifies how an integrated model that already contains a resource in the target component (BSS model part) is extended by a further resource node with the same name in the source component, which is therefore also related with the existing resource. The upper part of the figure shows the rule in abstract syntax and with explicit left and right hand sides. For all following rules, we use the compact notation (bottom part) based on the concrete syntax. In this case, the left and right hand sides are shown as one graph and the right hand side only elements are marked by double plus signs and additionally by green line colour. In order to initially create resources in the target component the rule "Resource2Resource" in Fig. 6.5 is used, where the additional negative application condition (NAC1) ensures that the BSS model part does not already contain a resource with the same name. In the same way as for resources there are two rules for synchronously creating and relating the actors between the source and target component, but the rules are given in abstract syntax, because there are different visualizations for actors (for persons and applications).

The second part of triple rules for the model transformation $Process2Service$ contains the rules "readAccess" and "writeAccess", which do not satisfy the standard termination criterion for forward transformations, which requires that each TGG-triple rule is creating on the source component. However, we can show termination also in this case by Fact. 6.2.4 using the more general termination result given by Thm. 5.2.14 in Ch. 5.

Example 6.2.2 (Triple Rules for $Process2Service$ (Second part))**.** *Further TGG-triple rules for the model transformation $Process2Service$ are shown in Fig. 6.6. Note that some rules do not specify all attributes if they are not necessary for the matching. Rule "Function2Channel" synchronously creates a function node and the corresponding channel. The next rule "execute" creates a link between an actor and the function he executes in the source component and it creates a connection between the corresponding channel node and the involved actor in the target component, which specifies the permitted communication at this function. The following four rules ("input", "output", "atResource" and "directCommunication") create data elements and the corresponding connections in the source component and do not change the target component. Intuitively, they prepare for the translation of these elements by the subsequent rules, because the interrelation of these elements with the BSS model elements is more complex. The rule "readAccessDirect" as well as "readAccess" establish new links in the target component specifying read access of actors to resources, if such a link is not present already, which is ensured by the NACs. Similar to the rule "readAccess" there is the rule "writeAccess" (not depicted), which requires a data flow in opposite direction and creates a link for write access ("w"). Even though the last three rules are not source creating, termination of the derived model transformation we will show termination using the extended criterion given by Thm. 5.2.14.*

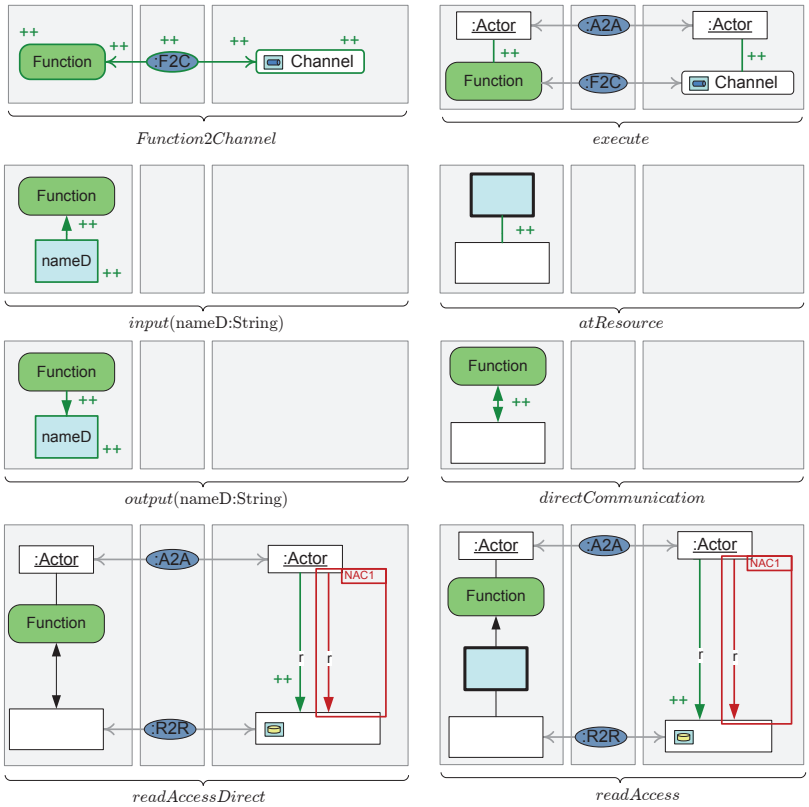


Figure 6.6: Further triple rules (second part)

Note that the attribute values for the node names in rule “readAccess” are not specified. This simplifies the rule. Indeed, the attribute values are not necessary and this way, one single NAC is sufficient. If the rule would additionally contain variables for the names then – in order to ensure termination – the rule would also contain further folded NACs of $NAC1$, where some of the variables are equal, i.e. identified by the folding. Without the additional NACs the forward translation rules could be applied at non-injective matches (identified data values) arbitrary often. The tool AGG provides a user friendly feature in this context. NACs are always interpreted to by possibly further folded on the elements that also occur in the left hand side of the rule, i.e. the user does not have to explicitly specify

these foldings. A formal specification of the induced folded NACs and NAC consistency checks is presented in [KHM06].

As presented in Ch. 5 model transformations based on TGGs are executed via the derived operational rules. In our case study we present one derived source rule and the corresponding forward rule. The complete model transformation was developed using the tool AGG based on the derived forward translation rules, which extend the forward rules by additional translation attributes for embedding the effect of the source rules as described in Sec. 5.1.2.

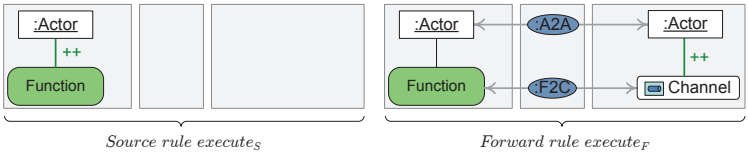


Figure 6.7: Derived source and forward rules for the triple rule "*execute*"

Example 6.2.3 (Operational Rules for *Process2Service*)**.** *Figure 6.7 shows the derived operational rules for the TGG-triple rule "execute" in Fig. 6.6. The source rule – denoted by the subindex "S" – is obtained by restricting the rule to the source component and it is used for parsing a given source model. The forward rule – denoted by the subindex "F" – does not change the source component and completes the missing elements in the correspondence and target component, which is just the link between the actor and the channel for the forward rule "execute$_F$".*

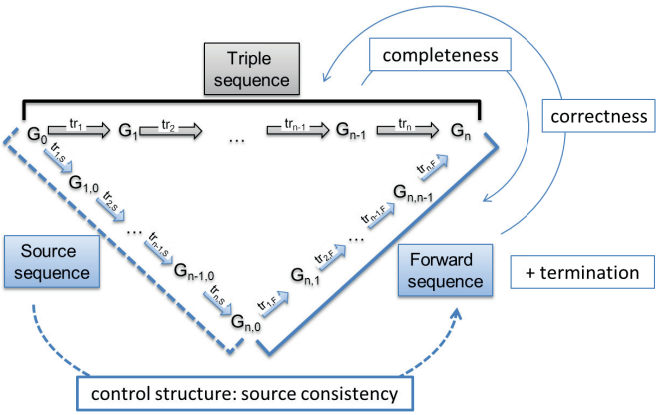


Figure 6.8: Correctness and completeness results for TGGs

Based on the developed model transformation techniques in Ch. 5 and the derived operational rules for the TGG of our case study, we can now execute the model transformation. Therefore, we can also apply the formal results shown in Ch. 5 and we will apply in particular the results concerning syntactical correctness, completeness and termination. For this reason, we briefly review these results from an intuitive point of view.

Syntactical correctness means that the resulting target model G_T of a performed model transformation corresponds to the given source model G_S according to the language of integrated models that is generated by the TGG-triple rules. More precisely, there is an integrated model $(G_S \leftarrow G_C \rightarrow G_T) \in VL = \{G \mid \varnothing \Rightarrow^* G$ via triple rules $TR\}$.As shown in Fig. 6.8 this property is ensured by the control structure *source consistency* together with termination, which ensures that for each source consistend forward transformation there is a corresponding source sequence, such that both sequences can be composed leading to a triple graph transformation sequence which synchronously builds up the source and target models. Vice versa, completeness ensures that for each integrated model $(G_S \leftarrow G_C \rightarrow G_T) \in VL$ there is a source consistent forward transformation sequence, which means in particular that the model transformation can be performed for any valid source model.

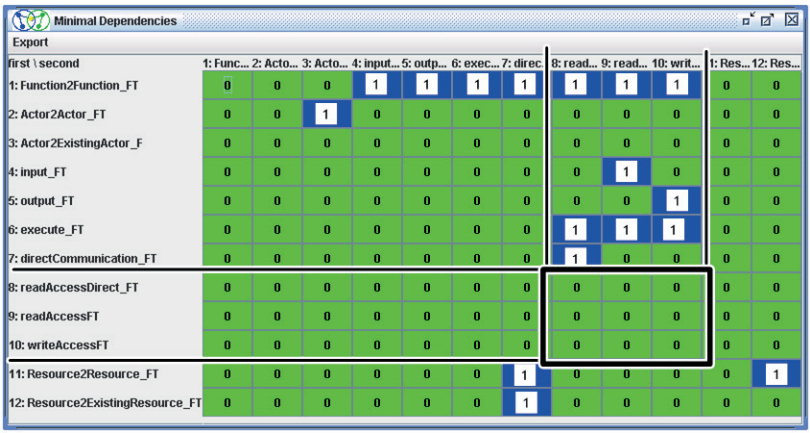


Figure 6.9: Dependencies between the transformation rules, generated by AGG

Fact 6.2.4 (Termination of $Process2Service$)**.** *The model transformation $Process2Service$ based on forward translation rules terminates for each finite business process model $M_1 \in VL_S$.*

Proof. We apply Thm. 5.2.14 of Ch. 5 by dividing the rule into the subsets $TR_{FT,1} = \{readAccess_{FT}, readAccessDirect_{FT}, writeAccess_{FT}\}$ and $TR_{FT,2} = TR_{FT} \setminus TR_{FT,1}$. First of all, the rules in $TR_{FT,2}$ are source creating. It remains to show that all rules in $TR_{FT,1}$ are self-disabling and pairwise independent if all NACs are removed. Each rule $tr_{FT,1} \in TR_{FT,1}$ has a NAC $NAC1$. The shift construction $\text{Shift}(l_{FT,1}, (r_{FT,1}, R_{FT,1})$ yields exactly this NAC, because $l_{FT,1}$ is the identity and the right hand side contains only one additional element, which is an edge. Furthermore, there are no further possible foldings of $NAC1$ via almost injective matches, i.e. matches that are injective on the graph part, because the types of the elements are different. Therefore, the rules are self-disabling. All three rules are non-deleting and only create edges of types that do not occur in the left hand side of one of the rules. Therefore, they are independent if NACs are neglected, which can be seen as well at the dependency matrix in Fig. 6.9 generated by AGG, where no combination of rules in $TR_{FT,1}$ shows a dependency. Therefore, $Process2Service$ is terminating. □

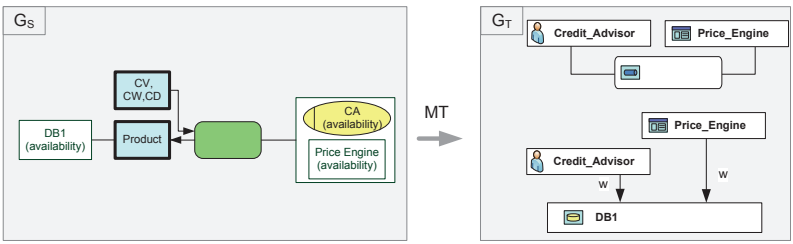


Figure 6.10: Model Transformation with source model G_S and target model G_T

Example 6.2.5 (Model Transformation)**.** *Fig. 6.10 shows a source model G_S and the derived target model G_T by applying the forward translation rules of the triple graph grammar TGG for the model transformation $Process2Service$. The credit advisor uses the price engine at the same function. For this reason, both actors are connected via a communication channel in the resulting BSS model. Furthermore, both actors create some output data and store it on the data base "DB1". For this reason, the generated BSS model specifies the necessary write access rights for both actors.*

In order to improve intuition, the icons for the different types of actors (external person, employee and application) are used in the concrete syntax, but the distinction is not derived from the model transformation but from the modelled inheritance relation in Fig. 6.2.

The model transformation sequence $(G_S, G'_0 \overset{tr^*_{FT}}{\Longrightarrow} G'_n, G_T)$ *consists of the following steps* $G'_0 \overset{Actor2Actor_{FT}}{\Longrightarrow} G'_1 \overset{Actor2Actor_{FT}}{\Longrightarrow} G'_2 \overset{Function2Channel_{FT}}{\Longrightarrow} G'_3 \overset{execute_{FT}}{\Longrightarrow} G'_4 \overset{execute_{FT}}{\Longrightarrow} G'_5 \overset{input_{FT}}{\Longrightarrow} G'_6 \overset{input_{FT}}{\Longrightarrow} G'_7 \overset{input_{FT}}{\Longrightarrow} G'_8 \overset{output_{FT}}{\Longrightarrow} G'_9 \overset{atResource_{FT}}{\Longrightarrow}$

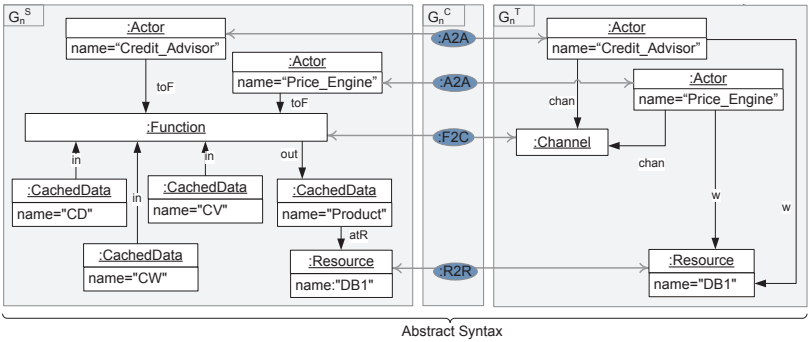


Figure 6.11: Resulting triple graph G_n of the model transformation in abstract syntax

$G'_{10} \xRightarrow{writeAccess_{FT}} G'_{11} \xRightarrow{writeAccess_{FT}} G'_{12}$. *The triple graph* G'_{12} *after removing the translation attributes is shown in Fig. 6.11 in abstract syntax and the triple graph contains the resulting target model* $G_T = G_n^T$ *in its target component as shown in concrete syntax in Fig. 6.10.*

6.3 Analysing Conformance of Business Process to Business Service Models

Business service structure models specify available resources and certain structures which constrain the execution of business processes. For this reason, business process models have to conform to the given business service structure models. This section presents formal techniques and results based on Ch. 5 which support the analysis of conformance of business process models and we apply these techniques to the case study $Process2Service$.

At first, we consider the basic case, where models of different domains are completely interrelated, i.e. we can define a triple graph grammar for integrated models containing complete domain models. By Thm. 6.3.3 we show that conformance can be checked with complete automation in this case by applying the corresponding model transformation. Furthermore, by Cor. 6.3.4 we can generate business service models out of business process models, such that conformance is ensured. This is an important result from the application point of view, because the generated business service structure models can be used as basis for further refinement, which substantially improves the efficiency of modelling in the business service structure domain. In several cases, the domain models are only partially interrelated. For this reason, we also consider restricted conformance (Def. 6.3.6) for which

we can directly apply the previous results, which allows us to show restricted conformance for our case study.

Standard conformance with respect to a given triple graph grammar is formalized as follows. Given a model M_1 of the source language and a model M_2 of the target language then M_1 conforms to M_2 if there is an integrated model containing M_1 as source model and its corresponding target model M_2.

Definition 6.3.1 (Conformance)**.** *Let $TGG = (TG, \emptyset, TR)$ be a triple graph grammar. A model $M_1 \in VL_S$ in the source language* conforms *to a model $M_2 \in VL_T$ in the target language iff there is an integrated model $(M_1 \leftarrow M^C \rightarrow M_2) \in VL$.*

However, in some cases, like in our case study, the given source models usually do not contain all necessary information for deriving the target model (BSS model). In particular, WDEPC models do not provide information about the inheritance relation within BSS models. For this reason, we generalize the notion of conformance to *forward conformance*, which does not require that the model M_2 is a model of the target language VL_T, but only typed over TG^T. Forward conformance is satisfied, if there is a model M_2' that corresponds to the given model M_1 via the TGG, such that M_2' can be mapped to M_2 via a conformance morphism. For this reason, we distinguish the special class of Morphisms $\mathcal{CON}$, which contains all conformance morphisms and at least all identity morphisms.

Definition 6.3.2 (Forward Conformance)**.** *Let $TGG = (TG, \emptyset, TR)$ be a triple graph grammar and $\mathcal{CON}$ be the class of conformance morphisms. A model $M_1 \in VL_S$ in the source language* forward conforms *to a model M_2 w.r.t. TGG and $\mathcal{CON}$ iff M_2 is typed over TG^T and there is a model $M_2' \in VL_T$, such that M_1 conforms to M_2' and there is a* conformance morphism*: $m : M_2' \rightarrow M_2$.*

In our case study, the class $\mathcal{CON}$ of conformance morphisms consists of morphisms that are identities on data values, i.e. attribute values are preserved. The explicit definition of $\mathcal{CON}$ is given in Ex. 6.3.5.

Based on Def. 6.3.2 and the formal results in Ch. 5 for model transformations we show by Thm. 6.3.3 below that the analysis of conformance can be performed by executing the corresponding model transformation and performing pattern matching for the derived corresponding model M_2' into M_2. An example of our case study, where we apply Thm. 6.3.3 is given by Ex. 6.3.7 for the more general notion of restricted forward conformance (see Def. 6.3.6).

Theorem 6.3.3 (Forward Conformance)**.** *Let $TGG = (TG, \emptyset, TR)$ be a triple graph grammar with derived model transformation $MT : VL_S \rightarrow VL_T$ based on forward rules and let $\mathcal{CON}$ be the class of conformance morphisms. Then, a model $M_1 \in VL_S$ forward*

conforms to a model M_2 typed over TG^T w.r.t. TGG and $\mathcal{CON}$ iff there is a model transformation sequence $(M_1, G_0 \overset{tr_F^*}{\Longrightarrow} G_n, M_2')$ via MT for which a conformance morphism $m : M_2' \to M_2$ exists.

Proof. Direction "$\Leftarrow$"
Since M_1 conforms to M_2 we can conclude according to definition Def. 6.3.1 that there is a target model $M_2' \in VL_T$ with $(M_1 \leftarrow M^C \rightarrow M_2') \in VL$ and there is a conformance morphism $m : M_2' \to M_2$. By Thm. 5.2.1 we know that MT is complete and therefore, the model transformation sequence $(M_1, G_0' \overset{tr_F^*}{\Longrightarrow} G_n', M_2')$ exists.
Direction "$\Rightarrow$"
Since MT is syntactically correct by Thm. 5.2.1 the model transformation sequence $(M_1, G_0' \overset{tr_F^*}{\Longrightarrow} G_n', M_2')$ implies that $(M_1 \leftarrow M^C \rightarrow M_2') \in VL$. Thus, the existence of the conformance morphism m implies that M_1 conforms to M_2 according to definition Def. 6.3.1. □

As a direct consequence of Thm. 6.3.3 we have by Cor. 6.3.4 that each model transformation with functional behaviour generates target models to which the given source model conform to. This allows for an agile development process, where business process models are developed before all details of the corresponding business service structure models are known and specified. The developed business process models can be transformed to business service structure models, which are then used as basis for further refinement.

Corollary 6.3.4 (Sound Completion)**.** *Let $TGG = (TG, \emptyset, TR)$ be a triple graph grammar with derived model transformation $MT : VL_S \to VL_T$ based on forward rules and let $\mathcal{CON}$ be the class of conformance morphisms. Let $M_1 \in VL_S$ be a source model and let $(M_1, G_0 \overset{tr_F^*}{\Longrightarrow} G_n, M_2)$ be a model transformation sequence via MT. Then, M_1 forward conforms to $M_2 \in VL_T$ w.r.t. TGG and $\mathcal{CON}$.*

Proof. We use the conformance morphism $m = id : M_2 \to M_2'$ for $M_2' = M_2$. By Thm. 6.3.3 we then have that M_1 conforms to M_2. □

The concept of node type inheritance is used in several domain specific languages for structure models in order to reduce the size of models and to improve conciseness. This is the case, for instance, in UML class diagrams [OMG10] or the business service structure models of our case study. From the formal point of view, this means that we consider the category of graphs with inheritance [EEH09], called **IGraphs**. This means that checking whether a business process model M_1 conforms to a business service structure model M_2 requires to validate the existence of an IGraph morphism $f : M_1 \to M_2$, which does not necessarily have to be a pure graph morphism. Instead, the notion of IGraph morphisms generalizes graph morphisms in the sense that source and target nodes of an edge e can also be mapped to descendent of the source and target nodes of the image $f_E(e)$ of the edge.

This more general form of morphism allows for a suitable notion of conformance, because the communication channels and access rights used by a business process model M_1 may be generally defined in the corresponding service structure model M_2 for node types that occur higher in the hierarchy.

Remark 6.3.5 (Graphs with Inheritance)**.** *Graphs with inheritance are given by a pair $IG = (G, I)$ consisting of a graph G and an inheritance relation $I \subseteq V_G \times V_G$ defining the parent-children relations between the node V_G of G. The clan of a node contains all its descendants according to I, i.e. $clan(n) = \{n' \in V_G \mid (n', n) \in I^*\}$ with I^* being the reflexive and transitive closure of I. Morphisms in* **IGraphs** *are general clan morphisms, i.e. an IGraph morphism $f : IG = (G, I) \to IH = (H, I)$ given by a mapping $f = (f_V, f_E) : G \to H$ from a graph G to a graph H generalizes the notion of graph morphism. The node mapping f_V is required to map source and target nodes of an edge e to nodes in H which are soure and target nodes of $f_E(e)$ or some of their children, i.e. $f_V(source_G(e)) \in clan(source_H(f_E(e)))$ and $f_V(target_G(e)) \in clan(target_H(f_E(e)))$.*

As mentioned before, there are some elements of the WDEPC models in Ch. 4, which are not relevant for checking conformance w.r.t. a BSS model and therefore, we first reduce the WEPC model by a restriction to the relevant types. This idea leads to the general notion of restricted forward conformance, where a model M_1 typed over TG_1 is restricted to a model $M_1|_{TG^S}$ typed over TG^S by removing all elements in M_1 that are not typed over TG^S. A type restriction $r_1 : TG^S \hookrightarrow TG_1$ specifies that TG^S is a subgraph of TG_1 and therefore, TG^S usually contains less types than TG_1.

Definition 6.3.6 (Restricted Forward Conformance)**.** *Let $TGG = (TG, \emptyset, TR)$ be a triple graph grammar and let $\mathcal{CON}$ be the class of conformance morphisms, $r_1 : TG^S \hookrightarrow TG_1$ be a type restriction and let M_1 be a model typed over TG_1. Then,* M_1 forward conforms *to $M_2 \in VL_T$ w.r.t. r_1, if the restricted model $M_1|_{TG^S}$ of M_1 is a model in VL_S and $M_1|_{TG^S}$ forward conforms to M_2 w.r.t. TGG and $\mathcal{CON}$.*

In order to analyse restricted conformance the given model M_1 is first restricted, which is performed as a pullback construction according to [EEEP09]. Intuitively, all elements whose types do not occur in the restriction type graph TG^S are removed as shown for our case study in Fig. 6.12. Based on the restricted model we can perform the model transformation and in the case of success we have by Thms. 5.2.1 and 5.1.14 that it is a model in VL_S. Thereafter, we can apply Thm. 6.3.3 to check for (complete) conformance to M_2. This way, the analysis is performed using completely automated techniques.

Example 6.3.7 (Conformance of Business Process Models)**.** *In order to check conformance of the WDEPC process model in Fig. 4.2 of Ch. 4 we first define the class $\mathcal{CON}$ of conformance morphisms. They are given by clan morphisms that are identities on the attribute values. This means that a clan morphism $m = ((m_V, m_D, m_E, m_{NA}, m_{EA}), m_A) :$*

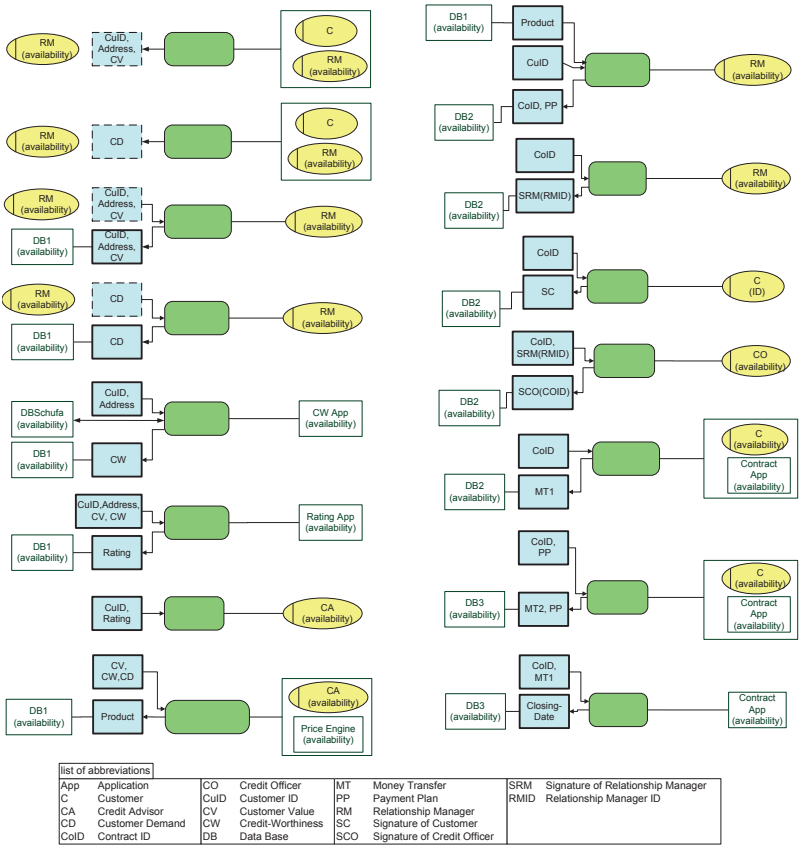


Figure 6.12: WDEPC process model after type restriction

$IG_1 \to IG_2$ is a conformance morphism, if the mapping m_D for data value nodes is the identity. (see Def. 3.1.2 and Def. 3.1.3 for attributed graph morphisms), i.e. the data value edges point to the same values in the image graph IG_2 as they do in the domain graph IG_1.

Recall, that we first extend the WDEPC model by attributes "name:String" to make the labels explicit as described before Ex. 6.2.1 in Sec. 6.2. Furthermore, we remove the resources of type "contract" to simplify the example and derive the WDEPC model M_1. If these resources are kept, however, then the corresponding BSS model would also contain them to provide conformance.

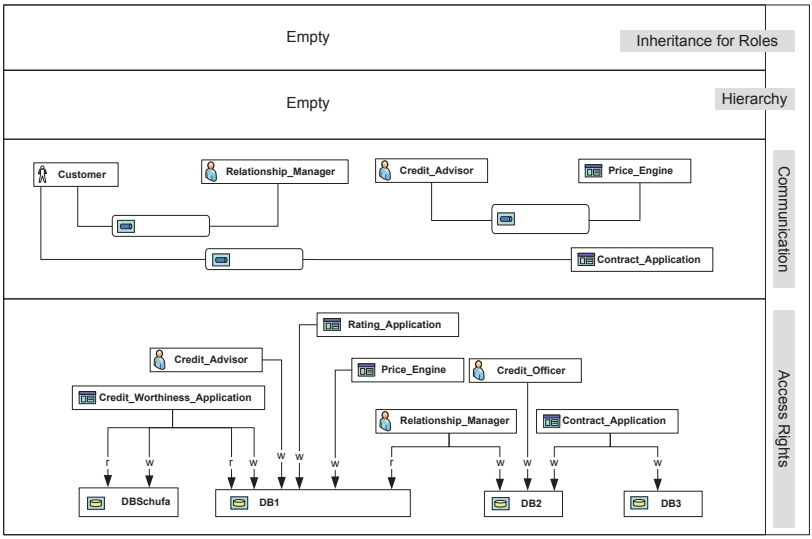


Figure 6.13: Generated business service structure model $M_2' \in VL_T$ (BSS model)

The type restriction is given by the morphism $r_1 : TG^S \to TG_{Meta}$ from the type graph TG^S of the triple graph grammar as shown in Fig. 6.3 into the meta type graph TG_{Meta} for WDEPC models in Fig. 4.1 of Ch. 4. Thus, e.g. the event nodes and adjacent edges are removed. The restriction of the WDEPC model M_1 in Fig. 4.2 leads to the restricted model $M_1|_{TG^S}$ shown in Fig. 6.12. The model transformation was executed using the tool AGG based on the derived forward translation rules of the triple graph grammar in Sec. 6.2 and the resulting target model is the generated BSS model M_2' in Fig. 6.13. The successful execution of the model transformation implies in particular that the restricted model $M_1|_{TG^S}$ is a model in VL_S.

As mentioned in Sec. 6.1 each actor in a BSS model has his own private communication channel and only the additional channels for communication with other actors are shown in concrete syntax. Furthermore, also mentioned in already in Sec. 6.1, nodes in a BSS model with the same name may occur multiple times in the concrete syntax but they refer to the same element in the abstract syntax as it is common praxis in visual modelling techniques like, e.g., UML [OMG10].

Now, there is a conformance morphism $m : M_2' \to M_2$ from the generated BSS model into the given BSS model in Fig. 6.2. In order to explain this mapping, the given BSS model M_2 is depicted again in Fig. 6.14. The mapping is given by the position in visual notation,

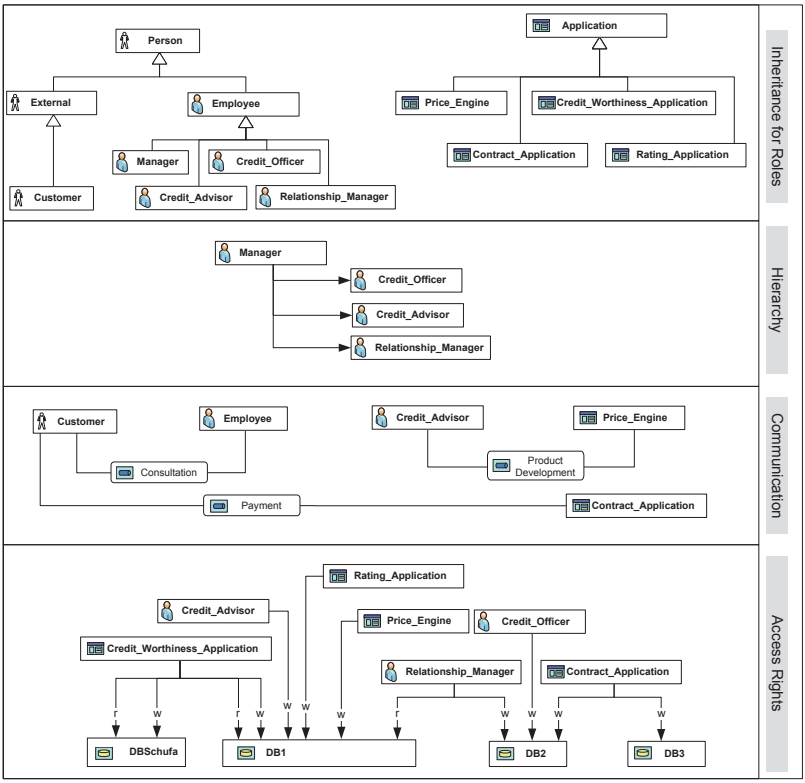


Figure 6.14: BSS model $M_2 \notin VL_T$ for conformance analysis

except for the node "Relationship_Manager" in the communication part. According to the conformance morphism condition we have that attribute values do not change, i.e. m *maps the node "Relationship_Manager" to the node "Relationship_Manager" and also the attribute value. Furthermore, according to the visual position, the conformance morphism maps the left channel node (between the customer and the relationship manager in the communication part) to the channel node "Consultation". This mapping is compatible with the inheritance relation, because the relationship manager is an employee. Note that the channel nodes in* M_2' *do not have name attributes, because this information is not available in the WDEPC models. But the missing attributes are not problematic for the conformance morphism* m*, because we use the flexible attribution concept based on E-*

Graphs [EEPT06], where attribute values of structure nodes may be undefined. Thus, the WDEPC model M_1 in Fig. 4.2 conforms to the BSS model M_2 in Fig. 6.14.

The conformance morphism in our case study is injective, but in the general case a conformance morphism does not have to be injective. In particular for our case study, we can consider a process model which induces a communication channel between the relationship manager and the customer as well as between the credit officer and the customer. In this case, both channels are mapped to the same channel "Consultation"in M_2.

Finally, there are usually several WEDPC process models and all of them are required to conform to a given business service structure model. For this purpose, we can extend the notion of conformance by Def. 6.3.8, which effectively requires that each WDEPC model of a given set has to conform separately to the given BSS model in order to satisfy combined conformance. Moreover, we can therefore check WDEPC models in combination with their continuity snippets for combined conformance.

Definition 6.3.8 (Combined Conformance)**.** *Let $TGG = (TG, \emptyset, TR)$ be a triple graph grammar and let $\mathcal{CON}$ be the class of conformance morphisms, $r_1 : TG^S \hookrightarrow TG_1$ be a type restriction and let S_1 be a set of visual models typed over TG_1. Then, S_1* forward conforms to *M_2 w.r.t. r_1, if each $M_1 \in S_1$ forward conforms to M_2 w.r.t. r_1.*

The analysis of combined conformance is performed as for restricted forward conformance before, but iteratively for each model in the set of given process models.

6.4 Optimization of Conformance Analysis

The presented techniques for checking conformance of WDEPC process models to BSS service models are performed using the model transformation techniques of Ch. 5. In this section we briefly discuss possible optimizations, which are partly based on the results in Ch. 5 as well as Ch. 3 and partly concern the specific nature of the conformance analysis in possible application domains.

Conformance checks during the modelling phase should ensure quick response times in order to provide adequate usability and fast conflict resolutions. For this reason, the applied model transformation techniques have to be efficient. As shown by Thm. 5.2.27 we can ensure that a model transformation based on forward translation rules with functional behaviour does not require backtracking. Therefore, the execution is ensured to have polynomial time complexity, which is a good result in this context.

Theorem 6.4.1 (Efficient conformance check)**.** *Let $TGG = (TG, \emptyset, TR)$ be a triple graph grammar with derived model transformation $MT : VL_S \to VL_T$ based on forward translation rules TR_{FT} (Def. 5.1.28), such that all significant critical pairs of TR_{FT}*

are strictly confluent according to Def. 5.2.16. Let $\mathcal{CON}$ be the class of conformance morphisms, $r_1 : TG^S \hookrightarrow TG_1$ be a type restriction and let M_1 be a model typed over TG_1 with $M_1|_{TG^S} \in VL_S$ being the restricted model of M_1. Then, M_1 forward conforms *to $M_2 \in VL_T$ w.r.t. r_1, if the model transformation of the restricted model $M_1|_{TG^S}$ leads to a model M_2' for which a conformance morphism $m : M_2' \to M_2$ exists. Moreover, the execution of the model transformation does not require backtracking.*

Proof. By Thm. 5.2.27 we know that strict confluence of all significant critical pairs ensures functional behaviour and the model transformation does not require backtracking. Functional behaviour of MT ensures termination for each source model $M \in VL_S$ and unique results for the execution of MT. Thus, the model transformation can be executed for $M_1|_{TG^S} \in VL_S$ leading to a model transformation sequence $(M_1|_{TG^S}, G_0' \overset{tr^*_{FT}}{\Longrightarrow} G_n', M_2')$ with unique result M_2'. By Thm. 6.3.3 we can therefore conclude that M_1 conforms to M_2 iff a conformance morphism $m : M_2' \to M_2$ exists. □

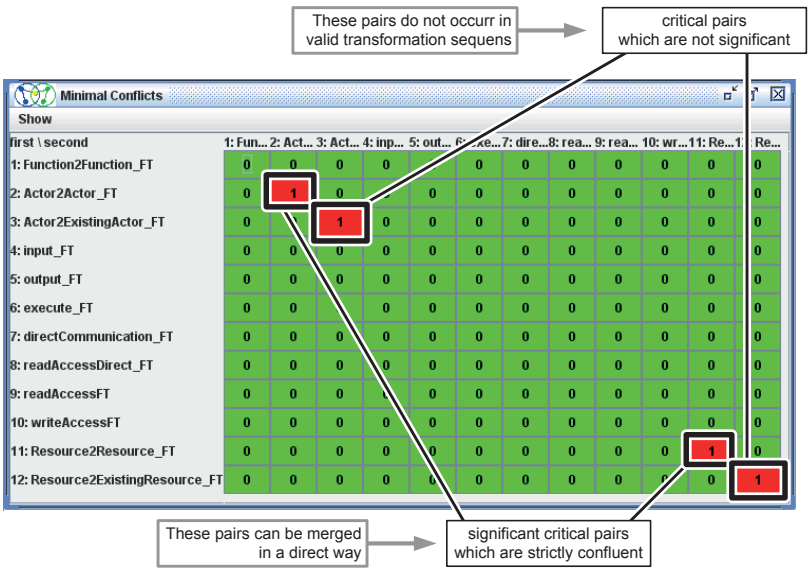


Figure 6.15: Critical pair analysis with the tool AGG

Example 6.4.2 (Efficient Conformance Check)**.** *In order to analyse the critical pairs of the forward translation rules using the tool AGG we flatten the forward translation rules to*

plain rules according to Def. 5.2.22. Based on the critical pair analysis engine of the tool AGG we verified that the significant critical pairs are strictly confluent. Critical pairs are significant, if they can be embedded in valid transformation sequences, i.e. transformation sequences that start at valid source models.

The amounts of generated critical pairs by AGG are listed for each combination in Fig. 6.15. The pairs with same rule and match are already filtered out, because in these cases the pairs $(P_1 \xleftarrow{p_i,m_i} K \xrightarrow{p_i,m_i} P_1)$ *are strict confluent by the empty transformations* $(P_1 \Rightarrow^* P_1 \ ^*\Leftarrow P_1)$*, i.e. no rule is applied. Moreover, 4 non-significant critical pairs were automatically filtered out by AGG using the well-formedness constraints of the language of WDEPC models given in Sec. 4.1.1, which were formalized as graph constraints. In addition to that, AGG filtered out 44 critical pairs that are strictly confluent in the following direct way. The critical pairs are of the form* $(P_1 \xleftarrow{p_i,m_i} K \xrightarrow{p_j,m_j} P_1)$*, i.e. both rules have the same effect and they create elements at the same context and preserve the common parts. Therefore, we derive strict confluence by the empty transformations* $(P_1 \Rightarrow^* P_1 \ ^*\Leftarrow P_1)$*, i.e. no rule is applied.*

The remaining critical pairs are shown in the generated table of AGG in Fig. 6.15. They concern the rules p_2 *="Actor2Actor*$_F$*",* p_3 *="Actor2ExistingActor*$_F$*",* p_{11} *="Resource2Resource*$_F$*", and* p_{12} *="Resource2ExistingResource*$_F$*" with combinations* (p_2, p_2)*,* (p_3, p_3)*,* (p_{11}, p_{11})*, and* (p_{12}, p_{12})*. The combinations* (p_3, p_3) *and* (p_{12}, p_{12}) *contain overlapping graphs, which show either two equally named resources or two equally named actors in the target component. This situation cannot occur, because the only creating rule for resource nodes* (p_{11}) *and the only rule for creating actor nodes* (p_2) *in the target component ensure by their NAC that no equally named node is present already and the target component is empty at the beginning of the transformation. Therefore, these pairs are not significant since they cannot be embedded in transformation sequences starting at a valid source model. Furthermore, the other two critical pairs are strictly confluent. For* (p_2, p_2) *the merging steps are* $P_1 \xrightarrow{p_3,m_3} K'$ *and* $P_2 \xrightarrow{p_3,m_4} K'$ *and rule* p_3 *does not have any NAC. Analogously, for* (p_{11}, p_{11}) *the merging steps are* $P_1 \xrightarrow{p_{12},m_3} K'$ *and* $P_2 \xrightarrow{p_{12},m_4} K'$ *and rule* p_{12} *does not have any NAC.*

Therefore, we have by Thm. 5.2.27 that MT *has functional behaviour and does not require backtracking.*

If however, functional behaviour cannot be ensured, but termination is ensured, we can perform backtracking and calculate all valid target models to a given source model. This process can be improved with respect to efficiency by iteratively constructing a corresponding subobject transformation system as described at the end of Sec. 5.3.2. By Def. 6.3.1 we have that the existence of one corresponding model M_2' with conformance morphism is sufficient. Thus, the backtracking procedure can be stopped as soon as such a model is derived, because we have by the correctness result (Cor. 5.2.6) for model transforma-

tions based on forward translation rules that each resulting target model is part of a valid integrated model that also contains the given WDEPC model.

Finally, concerning combined conformance, the efficiency of the analysis can be improved in cases, where models are stored in data bases, which is a common praxis in distributed development processes. Furthermore, we assume that also the model transformation techniques are implemented as data base operations, which is already planned in a current research project. In this case, the WDEPC process models is first combined to a single process model by disjoint union. Thereafter, the conformance check is performed for the combined model. This approach usually increases efficiency, because only one single data base query has to be sent in order to check for conformance. This means that the respond times for sending the queries to a server are reduced to only one and indeed, these response times are usually very significant concerning the overall efficiency.

Summing up, the provided model transformation techniques allow for efficient conformance checks during the development process and further improvements can be achieved for data base driven model repositories.

Chapter 7

Prototypical Tool Support: AGT-M

During the study of the various problems occurring in enterprise modelling we developed the prototypical tool AGT-M (algebraic graph transformation based on Mathematica), which shows the applicability and efficiency of the developed formal techniques of Chapters 3 to 6. Furthermore, we used some highly optimized components of the already existing tool AGG [AGG10] (attributed graph grammar system) for specific tasks. This chapter presents an overview of the package structure of the provided tool support and compares the main features as well as capabilities with those of other available graph transformation tools.

7.1 AGT-M: Algebraic Graph Transformation Based on Wolfram Mathematica

During the evaluation of existing transformation engines, in particular the tool AGG, we decided in cooperation with the project partners to provide a separate prototypical tool for the new developed techniques instead of extending an existing tool. The main reasons are the following. The overall structure of AGG has already become quite complex and the transformation engine uses optimized constructions, where morphisms and objects are not completely created and only the current host graph is stored during the transformation. However, the developed analysis techniques in Ch. 3 – also applied in the subsequent chapters – are based on the explicit construction of morphisms and graphs during the execution of a transformation sequence. Wolfram Mathematica [Wol10, Don09] provides optimized graph libraries and enables compact and fast implementations of mathematical constructions. Indeed, the developed implementation shows a one to one correspondence between the formal constructions of the DPO approach [EEPT06] and the implemented functions including the explicit construction of morphisms. Furthermore, AGG is based on Java which restricts the the maximal graph size to the available memory of the system. The

project partners, however, asked for more scalable systems that can additionally provide distributed execution engines. Wolfram Mathematica already provides several libraries for these requirements, such that AGT-M can be extended to scalable, parallel and distributed concepts based on the existing libraries in a later stage. The main components of AGT-M and the used components of AGG are visualized in Fig. 7.1.

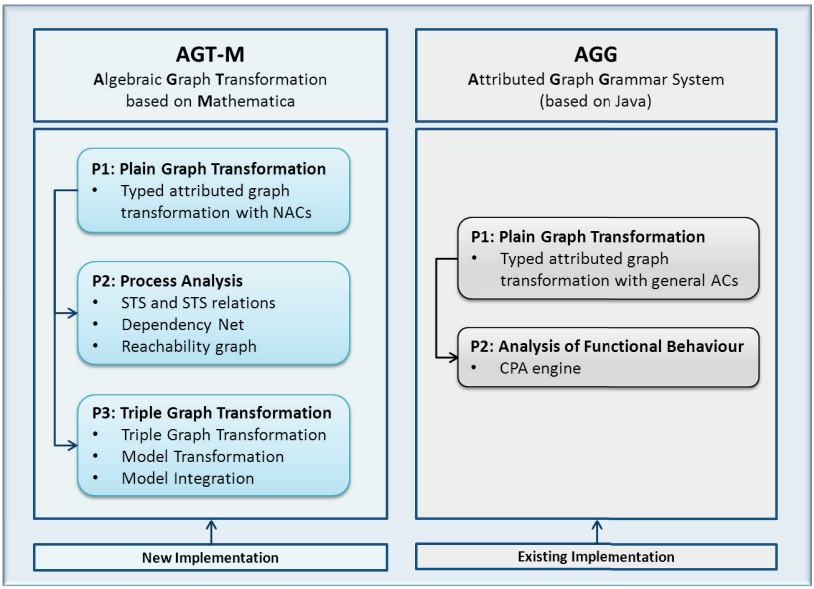


Figure 7.1: Overview of Tool Support

Package P1 is the basis of AGT-M and provides a transformation engine for plain graph transformations, which is presented in detail in [Ada10]. This engine supports the transformation of typed attributed graphs including the concept of node type inheritance according to [EEPT06]. This way, rules may contain abstract node types, which improves the modelling capabilities. Moreover, negative and positive application conditions are supported, which restrict the possible matches of a rule. The implementation is directly based on the formal definitions for typed attributed graph transformation using the double pushout approach, such that all graphs and morphisms that are involved in the transformation steps are created and thus, they are available for further formal analysis.

Package P2 of AGT-M provides the process analysis engine. This includes the construction of the subobject transforamtion system $STS(d)$ according to Ch. 3 for a given graph transformation sequence d that is executed by the transformation engine of package P1.

Based on this STS the engine provides the analysis of permutation equivalence of transformation sequences. For this purpose, at first the STS relations are generated and in a next step the dependency net $DNet(d)$ is generated. This dependency net is a standard place transition Petri net and specifies all possible equivalent executions of a given one. Finally, the generation of the reachability graph provides the explicit possible equivalent executions. Moreover, these components provide the bases for the analysis of business process models with respect to valid and equivalent business process runs, which additionally supports business continuity management according to Ch. 4. All these features of package P2 are applied for the case studies of Chapters 3 and 4.

The third package of AGT-M contains the triple graph transformation engine [Klj10]. This package uses the transformation engine of P1 and provides the execution of triple graph transformations according to the formal definitions in [Sch94, EEE+07]. This includes the generation of operational rules for model transformation (source rules and forward rules) as well as for model integration (source-target rules and integration rules) from a specified triple graph grammar. Thus, this package provides a formal triple graph transformation engine for model transformation and integration. According to the formal results concerning the analysis of model transformations based on TGGs we use the critical pair (CPA) analysis engine of AGG. Moreover, due to the shown equivalence of model transformations based on forward rules and those based on flattened forward translation rules we used AGG for executing the analysed optimized model transformations as presented in Ch. 5. In particular, AGG provides the analysis and verification of termination and functional behaviour. Moreover, AGG is used for the conflict detection of model transformation rules for which we can apply the conflict resolution results in Sec. 5.3.1. Finally, we applied AGG for analysing and checking conformance in Ch. 6.

We now compare the provided tool support by AGT-M together with the used components of AGG with other available tools concerning the main challenges of the research problems studied in this thesis.

7.2 Comparison of AGT-M with other Tools for Behaviour Analysis of Graph Transformation Systems

The behaviour analysis engine of AGT-M provides an efficient support for analysing permutation equivalence. This engine was successfully applied for the analysis of equivalent process runs with applications to mobile systems in Ch. 3 and business process analysis as well as business continuity management in Ch. 4.

There are some tools which provide capabilities for the verification of properties of graph transformation systems. The tool Groove [Gro10, Ren04b] provides a model checking engine for analysing the behaviour of a transformation system in general by generating the

complete state space, where isomorphic states are merged for improving efficiency. A further tool for the verification of properties of a graph transformation system is Augur [Aug10, KK08], which performs an over-approximation based on an unfolding of the system. This way, both tools are not appropriate for analysing equivalence for one particular execution, because all possible executions of the system are generated, where most of them are not relevant for the considered concrete execution. Furthermore, termination of the underlying graph transformation systems may be not ensured, such that in these cases certain bounds have to be specified manually.

The tool AGG also provides an engine for analysing graph transformation systems. Its CPA engine provides the generation and analysis of critical pairs for analysing possible conflicts and dependencies of rules. In general, there are many cases where rules can be applied in a dependent way, but for a given transformation sequence several concrete steps are often independent while the rules show possible dependencies. Therefore, the analysis of permutation equivalence using the CPA engine is not complete. Furthermore, the CPA analysis is complex, because it generates the complete set of all possible conflicts in an arbitrary context. The analysis of permutation equivalence with AGT-M instead uses the matches of the given transformation sequence, such that the costs of pattern matching are avoided. All together, the CPA engine of AGG is not adequate for analysing permutation equivalence.

Thus, the behaviour analysis engine of AGT-M fits best the requirements for tool support concerning the process analysis of graph transformation systems, i.e. the analysis of the equivalent concurrent behaviour for a concrete transformation sequence. Based on the execution of one particular run d of a graph transformation system the engine can generate the process model given by a subobject transformation system $STS(d)$, the derived dependency relations, the dependency net $DNet(d)$ with initial marking as well as its reachability graph. As shown in Ch. 3 this generation is efficient compared to a direct analysis of permutation equivalence and can be used for the analysis of equivalent business process runs and business continuity management as shown in Ch. 4.

7.3 Comparison of AGT-M with other Tools for Model Transformation

AGT-M provides a formally well founded implementation of model transformations based on triple graph grammars (TGGs), which strictly corresponds to the constructions presented in Ch. 5. All involved morphisms and objects during a TGG transformation are constructed and available for further analysis. This way, the formal results concering correstness and completeness are ensured and the formal results concerning termination, functional behaviour and information preservation can be applied for verifying these properties.

Furthermore, the tool can be directly extended for improving the efficiency of the execution of model transformations using the behaviour analysis engine according to Sec. 5.3.2. Moreover, the Mathematica environment provides highly optimized libraries including graph algorithms and libraries for grid computing, which ensures scalability of the implemented techniques.

Several plain graph transformation tools are available, like EMF Henshin [ABJ+10], the FUJABA tool suite [Fuj07], GrGen [grg06] or ATOM^3 [dV07], in which in-place model transformations can be specified, where the source model may be modified during the transformation till the resulting target model is derived. This additionally complicates the analysis and verification of termination. But in this case, the formal results available for triple graph grammars are not ensured, because the in-place rules usually do not correspond to operational triple rules and all control structures are manually specified. However, several case studies have been successfully performed using this in-place transformation approach [BE09, BESW11].

Some TGG implementations are available. The TGG extension for FUJABA [Fuj07] presented in [KW07] does not use the explicit formal notion of triple graphs, where the correspondence graph is connected to the source and target graphs via graph morphisms. Instead, the correspondence graph may contain elements that are not mapped to elements in the source or target graphs (violates left totality) and moreover, one correspondence element can be connected to more than one source element (violates right uniqueness). The FUJUBA based TGG tool MOFLON [TU 10, KKS07] solves this problem by restricting the editing of triple rules, such that each correspondence node is connected to exactly one source and one target element. Furthermore, the execution is controlled by iteratively updating a the set of translated nodes [SK08], which is intuitively similar to the notion of partial source consistency in Sec. 5.1.1, but there a formal proof showing that a successfully executed transformation ensures source consistency is missing up to now. However, source consistency concerns all elements, i.e. also attributes and edges and not only nodes, such that source consistency cannot be ensured by the algorithm in [SK08] for all valid source models, i.e. the algorithm does not guarantee completeness, even if backtracking is not necessary according to Sec. 5.2.2. Moreover, there is an extension for the tool Atom^3 [dV07, Gd06b], but again source consistency is not ensured.

Since the tools do not ensure source consistency, which is a sufficient and necessary control condition for correct and complete model transformations based on TGGs the results produced by these tools are not formally verified. Furthermore, in order to ensure efficient executions the tools usually do not perform any backtracking, such that for several valid input source models the tools will not be able to compute any target model as output result.

Fortunately, as presented in Sec. 5.1.2 correctness and completeness of model transformations based on TGGs can be ensured using the flattened version of the derived forward translation rules, where source consistency for complete sequences is ensured by the ad-

ditional translation attributes. This way, any graph transformation engine that soundly implements the DPO approach for plain typed attributed graph transformations [EEPT06] including a sound backtracking mechanism, can be used to perform correct and complete model transformations. Moreover, by Thms. 5.2.27 and 5.2.31 in Sec. 5.2.2 we can analyse TGGs using the tool AGG whether backtracking is required. This way, model transformations that do not require backtracking can also be performed on tools that do not support backtracking. Clearly, the usage of existing tools requires that the forward translation rules have to be derived first and it would be more user friendly to provide an additional graphical user interface for editing the triple graph grammar and generating the forward translation rules automatically.

Based on the formal results in Ch. 5 we have successfully analysed and executed model transformations based on TGGs for the case study in Ch. 5 using AGT-M and AGG and for the case study in Ch. 6 using the tool AGG.

Chapter 8

Related Work

In this chapter, we discuss on the one hand, how the developed techniques and results are placed within the area of graph and model transformation and how they are related to existing results in this area. On the other hand, we discuss and compare the presented solutions with related approaches concerning the support of enterprise modelling and business continuity management. The following Sections 8.1-8.4 discuss the related work concerning Chapters 3, 4, 5 and 6, respectively, while Chapters 2 and 7 include already related work.

8.1 Process Analysis of Graph Transformation Systems

Transformation systems based on the double pushout (DPO) approach [Roz97, EEKR99, EKMR99] with negative application conditions (NACs) [HHT96, EEPT06] are a suitable modelling framework for several application domains in the area of distributed and concurrent systems. The behaviour of these systems is formalized by an operational semantics given by a graph transformation system, where each transformation rule can simulate a step of the modelled system.

A process of a particular execution describes all possible equivalent executions. Correspondingly, a process of a transformation sequence via the rules of the operational semantics defines an equivalence class of transformation sequences. Processes of graph transformation systems based on the DPO approach are introduced in [CMR96] and characterized as occurrence grammars in [Bal00]. These concepts generalise the notion of processes available for the classical formal models in this context – namely Petri nets [Rei85] – which can be completely defined by restricted graph transformation systems (GTSs), while general GTSs are more expressive. In [BCH$^+$06], processes of graph transformation systems are lifted to the abstract setting of adhesive rewriting systems in order to generalise the process construction and analysis. This way, the analysis techniques can be instantiated

for transformation systems based on arbitrary adhesive categories [LS04], such as typed graphs, graphs with scopes and graphs with second order edges.

The concept and analysis of processes of transformation systems with negative application conditions (NACs) is more complex. As presented by the case study, the notion of switch equivalence with NACs is in general to strict for analysing equivalence of transformation sequences in this context. The reason is that switch equivalence with NACs based on sequential independence of transformation sequences with NACs [HHT96, LEO06, LEOP08, Lam09] does not relate all transformation sequences, which are intuitively equivalent in the way that they are switch equivalent without NACs and additionally, each transformation step satisfies the NACs of its rule. The reason is that several switchings of NAC-dependent steps have to be performed in order to derive a new transformation sequence that is again NAC-consistent, while the derived transformation sequences in between are not NAC-consistent. However, the derived new NAC-consistent transformation sequences are considered to be equivalent. Similarly, the notion of shift equivalence [Roz97, Kre86] for transformation sequences cannot be extended appropriately to the case with NACs, because it is also based on sequential independence of neighbouring steps. For this reason, we introduced the notion of permutation equivalence, which is by definition exactly the required equivalence relation as described above. Moreover, we extended the construction and analysis of processes to the more general framework of $\mathcal{M}$-adhesive transformation systems, such that the important category of typed attributed graphs is included. Furthermore, the developed techniques are extended to transformation sequences with general match morphisms, i.e. matches do not have to be injective as in the cases before.

The extended process construction leads to the new notion of subobject transformation systems (STSs), which generalise the concept of elementary Petri nets [RE96] being the class of process models for P/T Petri net processes, such that STSs form the class of process models for arbitrary $\mathcal{M}$-adhesive transformation systems. In order to provide a sound, complete and efficient analysis technique for permutation equivalence we presented how the dependency net for the given transformation sequence can be constructed, which purely specifies the dependencies between the transformation steps including the inhibiting effects of the NACs. Based on the reachability graph of the dependency net, all valid and equivalent permutations of the transformation sequence are derived. This way we have shown that the interleaving semantics of any $\mathcal{M}$-adhesive transformation system can be described using pure P/T Petri nets, which specify all equivalent permutations of the steps. The concrete transformation steps of the equivalent transformation sequences are derived from the firing sequence of the Petri net using the corresponding STS.

Some of the problems addressed in this paper are similar to those considered in the process semantics [KK04] and unfolding [Bal00, BKS04a] of Petri nets with inhibitor arcs, and actually we could have used some sort of inhibitor arcs to model the inhibiting effect

of NACs in the dependency net of a transformation sequence. However, we would have needed some kind of "generalised" inhibitor nets, where a transition is connected to several (inhibiting) places and can fire if at least one of them is unmarked. To avoid the burden of introducing yet another model of nets, we preferred to stick to a direct encoding of the process of a derivation into a standard marked P/T net, leaving as a topic for future research the possible use of different models of nets for our dependency net.

8.2 Reconfiguration in Business Continuity Management

The main purpose of business continuity is to provide continuity plans for the cases in which some failures occur during the execution of critical business processes in order to ensure business activity.

There are several approaches on adaptive processes which allow for dynamic changes of the process graphs. The ADEPT system [RD98] provides general reconfiguration operations and additional conditions that restrict the possible dynamic modifications during runtime in order to ensure that the modifications do not introduce inconsistencies. For example, the conditions ensure that new steps are only inserted as possible successor steps of the current state. In [RMRW08], the ADEPT system is extended by a trace based semantics for general change patterns for which the effect must be precisely defined. Moreover, the ADEPT system provides an interface for end users to perform general modifications of the process graph [DR09] and checks data dependencies before performing a transformation. A context-aware selection of options including preconfigured and checked process variants are available for the Provop approach presented in [HBR09], where users specify the dependencies between the available options by constraints. Further checks for basic semantic constraints concerning dependencies and mutual exclusions are performed in the SeaFlows project [LRD08].

However, our presented approach in Ch. 4 focuses on rather orthogonal aspects. Instead of providing general modification operations which are manually applied by some operational supervisors or actors within the process, our techniques are based on small continuity snippets and formalized functional and non-functional requirements from which we generate complete alternative process runs for different combinations of failures. These alternatives are already verified in advance in order to meet the formalized functional and non-functional requirements. For this reason, the active persons within the running business process do not have to manage and control all modifications and reconfigurations, but can concentrate on their business tasks. This reduces the risk of making incorrect ad hoc decisions. The actors can choose an alternative from a set of validated alternatives, which are possibly ranked by business values. Moreover, the procedure enables partial decisions, in the way that at first a subset of possible continuations is selected and the explicit

choice regarding the concrete continuation alternative is made at a later stage. All together, the efforts for modelling as well as management are kept low by providing an automated IT-support.

Concerning the analysis of business workflow models there are several studies on workflow patterns [vdAHKB03], which specify suitable model fragments for certain application scenarios as well as data-flow anti-patterns [TvdAS09], which specify substructures that may lead to inconsistencies including ambiguous data flow. The underlying process models are given by Petri nets or simple models consisting of action, event and control nodes. This means that only a few aspects of the business process models in Ch. 4 are captured. The data modifications are specified by inscriptions within the transitions of a Petri net (read, write and destroy operations). This way the patterns and anti-patterns support the detection of inconsistencies within models, but they cannot guarantee the correctness of them concerning, e.g., that the specified data manipulations can be effectively executed. Further analysis techniques for event driven process chain (EPC) models are presented in [Men07], which mainly focus on heuristics for the detection of possible deadlocks caused by inconsistent combinations of split and join control structures, like e.g. a combination of an *XOR* split with an *AND* join.

A more abstract way of modelling business processes is presented in [vdADG+08], where concrete configurations depending on the particular application scenario can be derived from a reference model. In this context, business process models are given by workflow nets, which are special Petri nets satisfying the following restrictive well-formedness criteria: (1) there is exactly one input place, (2) one output place, (3) all transitions and markings are reachable from the initial marking and (4) they are successors of the output place resp. final marking. The main result of this study shows that given a sound reference process model, then a configuration of this model which satisfies the workflow net conditions is sound as well, meaning that it can be completely executed and each action node can be used in at least one execution variant. While this approach is quite specific and does not concern the various side constraints like access rights to resources and additional security constraints, the general concept of reference models can be considered for further extensions of our techniques.

The above analysis techniques are mainly orthogonal to the generation techniques for business continuity management (BCM) based on extended EPCs and continuity snippets in Sec. 4.3.1, which ensure that the generated process runs can be executed, respect the data flow dependencies, and satisfy the additional functional and non-functional constraints. For this reason, there is a good potential that the above analysis techniques can be used to extend the presented BCM techniques. In particular, the general process reconfigurations available in the ADAPT Framework or presented for reconfigurable mobile ad hoc networks [PEH07] can be possibly combined with our approach in an orthogonal way.

8.3 Model Transformation

Model transformations are a key concept for the generation of domain and platform specific models and system components within model driven architecture (MDA, see [OMG01]) including the transformation concept QVT (query view transformations) [QVT08]. Apart from the usual focus of MDA on pure top-down generation strategies – like code generation – bidirectional model transformations are an important concept for integrating several domain specific languages and for supporting model based interoperability. A discussion on challenges and possible consequences for QVT transformations is given in [Ste10] including non-bijective model transformations and proposing triple graph grammars (TGGs) [Sch94, SK08] as one possible concept for further consideration.

Indeed, TGGs have been successfully applied with different purposes in a variety of domains [Gd06a, Gd06b, KW07, KS06, TEG+05]. Pattern-based model-to-model transformations have been introduced in [dLG08] and corresponding correctness, completeness and termination results have been presented in [OGdLE09], which are, however, limited in comparison with the results in this thesis. The triple rules for the triple language are itself generated from the modelled patterns and the presented notion of correctness is not symmetric, i.e. there is a distinction between forward and backward satisfaction. Moreover, the results consider NACs only for ensuring termination, but not for the specification of the triple language. The presented theory of model transformations based on forward rules has been extended in [GEH11, Gol10] from systems with NACs to systems with general application conditions called nested application conditions.

The developed techniques in Sec. 5.2.2 for analysing functional behaviour and information preservation of model transformations based on TGGs provide a systematic approach for analysing non-bijective model transformations and verifying several properties. Functional behaviour for a case study on model transformations based on "plain graphs" is already studied in [EEPT06] using also critical pair analysis in order to show local confluence. But the additional main advantage of our techniques for TGGs is that we do not have to require full confluence, but instead the weaker notion of translation confluence. Furthermore, we can ensure the strong results concerning correctness and completeness. A first approach to local confluence and functional behaviour for model transformations based on TGGs was already given in [EP08]. This approach, however, is very restrictive, because it is based on triple rules with kernels, which can be applied only to very simple examples.

Moreover, there is an enormous need for suitable formal analysis techniques for model transformations in order to detect inconsistencies, but also to ensure and verify the consistency of the enterprise model in the whole. Several existing model transformations in practice are encoded as XSLT-transformations, but a formal analysis technique concerning correctness and completeness is not available for them. Triple graph grammars have been used extensively for specifying model transformations and they provide a for-

mal foundation based on algebraic graph transformation [EEPT06], an intuitive visual notation and they show a high degree of maintainability. In comparison to plain graph transformation approaches for model transformation, like e.g. EMF model transformations [BET08, ABJ+10], TGGs provide formal results that can by analysed efficiently for quality assurance, e.g. concerning correctness and completeness. Furthermore, they facilitate bidirectional model transformations. Moreover, the operational rules including the necessary control structures are generated automatically from a single set of triple rules as presented in Sec. 5.1.2. This substantially reduces the modelling and specification efforts. The modeller purely specifies the interrelationship of source and target models by patterns, i.e. by defining how model fragments may be extended simultaneously on the source and target component. According to Sec. 5.3.3 we were able to master the major part of the main challenges of model transformations based on triple graph grammars as presented in [SK08, HHK10] concerning – in particular – correctness, completeness, termination, efficiency and expressiveness.

Finally there is a strong relationship with the model transformation algorithm in [SK08], which provides a control mechanism for model transformations based on TGGs by keeping track of the elements that have been translated so far at an intermediate step. In Sec. 5.1.1 we formalized the notion of elements that are translated at a current step by so-called effective elements and in Sec. 5.1.2 we have shown that the new translation attributes can be used to automatically keep track of the elements that have been translated so far.

8.4 Consistency Analysis of Heterogeneous Models

Enterprise modelling encompasses several different aspects, which are usually modelled by different heterogeneous types of models. These models are developed partly independent from each other in a decentralized way, where models are modified by different teams, who have specified knowledge in their domains. In order to analyse and ensure consistency between the different enterprise models, different approaches and techniques were developed.

A common strategy in software engineering for reducing the complexity of the consistency problem in whole is to define separate views [GEMT00, EEHT97] and to develop domain focussed models in these views that specify some partial aspects of the system. The integrated meta model of UML [OMG10] enables, on the one hand, the modelling in different visual notations using the different types of UML models and, on the other hand, specifies the syntactical overlappings between these models. Additional constraints (OCL and natural language) describe additional requirements that induce dependencies between the models. But this concept is too strict for enterprise modelling, where several different domain languages are used, which cannot be mapped directly to some UML model,

even if stereotypes are used for customization. Furthermore, decentralised modelling may introduce inconsistencies, which cannot be directly analysed and resolved.

The concept of views is generalised in [EEEP09] by providing a formal framework for views along type hierarchies given by hierarchies of meta models (resp. type graphs with constraints) in a flexible way. The assumption is that all partial models can be combined to form one integrated model typed over the integrated type graph, where the different modelling teams have only restricted knowledge about the complete meta model. Two models are considered to be consistent, if they are isomorphic on the common types with respect to the intersection of the domain type graphs. Further semantic consistency checks for the integration of viewpoint models are presented in [BKS04b].

However, this notion of consistency is still too strict for our scenario, where dependencies between heterogeneous models exist and have to be respected. The models usually do not coincide completely on the structure that is relevant for the dependencies. In particular, this is the case in our case study, where business process models may contain several fragments that have to conform to a single fragment of the corresponding business service structure model. Moreover, fragments of one domain have a different structure as the fragments in the other domain. For this reason, the main part of the dependencies between these models cannot be handled by the above concept based on an intersection of the type graphs.

In order to provide a general approach for analysing consistency between heterogeneous models we introduced the concept of conformance in Ch. 6, which is much more flexible, can be performed by automated formal techniques and allows for checking dependencies between models that do not necessarily share common substructures.

Summing up, there are several related techniques and approaches, which are partly focussed on different aspects. However, they do not provide the optimal solutions for the considered challenges and problems in the application domains of this thesis concerning the formal and automated support of decentralised enterprise modelling. Nevertheless, some concepts may be used for further extensions of the presented techniques.

Chapter 9

Conclusion and Future Work

In order to improve the capabilities and benefits of visual enterprise modelling and to solve main problems in this area we developed and presented new powerful and efficient techniques for the analysis and optimization of the involved models and the improvement of the modelling process based on new formal concepts and results in the area of graph and model transformation. This chapter summarizes and discusses the main contributions concerning the theoretical results from the general point of view (Sec. 9.1), the results for visual enterprise modelling (Sec. 9.2), the relevance for model driven software engineering (Sec. 9.3) and, finally, we discuss future work including further application domains (Sec. 9.4).

9.1 Summary of Theoretical Results

From the theoretical foundation point of view, this thesis provides new concepts and results, on the one hand, for behaviour analysis and concurrent semantics of $\mathcal{M}$-adhesive transformation systems and in particular for graph transformation systems as presented in Ch. 3. On the other hand, it provides new techniques and results for model transformations based on triple graph grammars as presented in Ch. 5.

Behaviour Analysis of Visual Languages Based on Graph Transformation

The general framework of $\mathcal{M}$-adhesive transformation systems provides an abstract view on many important instantiations, such as different types of graph transformation systems, and Petri net transformation systems - in particular for the modelling of workflows of reconfigurable mobile ad hoc networks [EHP+07, HEP07]. Each of these instantiations provides specific features relevant for their application domain.

The concurrent semantics of $\mathcal{M}$-adhesive transformation systems was studied up to now, only for the more restrictive setting of adhesive transformation systems [BCH+06] and

low-level Petri nets. The main contribution of this thesis in this context is a general notion of concurrent semantics for $\mathcal{M}$-adhesive transformation systems in the sense of an interleaving semantics together with suitable and efficient analysis techniques. This builds the basis for a future extension to a true concurrent semantics in this general framework, where several independent steps can be equivalently executed by one current step.

While negative application conditions (NACs) are an important and widely used control structure for $\mathcal{M}$-adhesive transformation systems the available results concerning the concurrent semantics of transformation systems do not consider NACs. Moreover, the standard notion of switch-equivalence based on sequential independence of neighbouring steps is too strict in the presence of NACs, i.e. there are equivalent transformation sequences with NACs which are not switch-equivalent with NACs as shown by the presented case study. For this reason, we introduced the new notion of permutation equivalence in Sec. 3.2 as a generalisation of switch equivalence with NACs. By definition, two transformations are permutation-equivalent, if they are switch-equivalent without NACs and moreover, each step of the transformation sequences satisfies the NACs. Therefore, all possible permutations of the transformation steps are considered.

Continuing, we presented the new concept of subobject transformation systems (STSs) in order to define the interleaving semantics of $\mathcal{M}$-adhesive transformation systems. They generalise the notion of occurrence grammars, which specify the interleaving semantics of adhesive transformation systems [BCH$^+$06]. In the case of low-level P/T Petri nets, they correspond to the notion of elementary nets. Furthermore, we extended the constructions to the case of transformation systems with general matching. This extension is important in the case of attributed graph transformation systems, where injective matching is too restrictive. Based on the new constructions we defined the process of a transformation sequence as a pair consisting of the derived STS and a relating mapping into the original $\mathcal{M}$-adhesive transformation system. As a first main result, we have shown by Thm. 3.4.12 that the analysis of the interleaving semantics of an $\mathcal{M}$-adhesive transformation system based on the new notion of processes is sound and complete. This means that we derive the set of all permutation-equivalent transformation sequences to a given transformation sequence d of an $\mathcal{M}$-adhesive transformation system by generating all legal sequences of rule names within the process of d. Moreover, we can guarantee an efficient construction of the process model according to Fact. 3.4.15, which states that the construction together with its dependency relations is performed in polynomial time with respect to the size of the given transformation sequence and its components.

In a further step, we presented an efficient analysis of the interleaving semantics of $\mathcal{M}$-adhesive transformation systems. This analysis is performed on a generated corresponding dependency net, which purely specifies the dependencies of a transformation sequence leaving out all concrete information about the involved objects. According to Thm. 3.5.3, the set of transition complete firing sequences of the dependency net exactly specifies the

set of permutation-equivalent transformation sequences to a given one. Therefore, the reachability graph of the dependency net, restricted to the transition complete firings, shows all permutation-equivalent transformation sequences. This way, we have shown that the interleaving semantics of an arbitrary $\mathcal{M}$-adhesive transformation system can be studied based on a fundamental model of concurrency – a marked low-level P/T Petri net.

Finally, we evaluated the efficiency of the analysis based on a convincing benchmark. The efficiency of the Petri net approach is based on two advantages. First of all, the constructed Petri net only specifies the dependencies among the steps of the derivation, ignoring the concrete structure of the involved graphs. Secondly, all NACs are respected already during the generation of the permutation-equivalent sequences.

Model Transformations Based on Triple Graph Grammars

Model transformation based on triple graph grammars [Sch94, SK08] have been applied in several application domains in model driven development and we have presented the main new results based on a quasi-standard case study in this area – namely the model transformation from class diagrams to relational data base models. The main advantages of model transformations based on TGGs are the intuitive specification, the bidirectional character, the automatic generation of operational rules, the formal foundation, the available expressive control structures like negative application conditions, the powerful results for analysis and optimization, and the preservation of the given source models in contrast to common in-place model transformation approaches [ABJ+10].

The main contributions for model transformations based on TGGs in this thesis are given by new powerful execution, analysis and optimization techniques, which ensure the fundamental properties of syntactical correctness and completeness. As presented in Sec. 5.3.3, the new developed concepts and results provide adequate solutions for most of the main challenges for model transformations based on TGGs according to [SK08] and according to the extended list of challenges in Sec. 5.1. In the following, we discuss the main results for these challenges concerning functional and non-functional requirements of model transformations.

Starting with the functional requirements of model transformations, we have shown that the presented model transformation approaches based on forward and forward translation rules in Ch. 5 ensure syntactical correctness and completeness. Both properties are defined with respect to the language of integrated models generated by the specified TGG, whose rules define how source and target models can be synchronously created. Syntactical correctness of a model transformation means that each executed model transformation starting at a valid source model generates a valid target model and, moreover, there is an integrated model containing both models. Completeness in this context means that each valid source model can be transformed into a target model, and vice versa, for each valid target model

there is a source model that can be transformed into this target model via the model transformation. By Thm. 5.2.1 we have shown that model transformations based on forward rules ensure syntactical correctness and completeness in general. Moreover, by Thm. 5.2.4 and 5.2.14 we provided sufficient conditions for ensuring termination of model transformations, which are very general and can be applied to many case studies. In particular, they suffice to show termination of the model transformations in our case studies.

Based on the new concept of forward translation rules we have shown new results for the analysis of functional behaviour, i.e. that the execution ensures unique results. While the general notion of functional behaviour of graph transformation systems is equivalent to the notion of confluence, this is not the case for model transformations, where not all transformation sequences are relevant, but only those which transform a valid source model into some target model and which are complete with respect to the execution strategy of the model transformation. And indeed, many model transformations are not generally confluent and therefore, the available techniques based on critical pairs [EEPT06, Lam09] for showing confluence cannot be applied. For this reason, we introduced the new notion of significant critical pairs and the concept of filter NACs, which filter out execution paths that would lead to a point for backtracking and additionally reduce the amount of significant critical pairs. By Thm. 5.2.27 we have shown that strict confluence of significant critical pairs ensures functional behaviour. This way we were able to show functional behaviour for the model transformations in Chapters 5 and 6. Moreover, we have shown by Thm. 5.2.31 that the absence of significant critical pairs ensures strong functional behaviour, which additionally ensures that the execution sequences of the model transformation are unique up to switch-equivalence.

In addition to the functional requirements, we provided further results concerning non-functional requirements. In particular, we presented a new technique for showing that a model transformation is (complete) information preserving. Standard information preservation requires that for each forward model transformation sequence there is also a backward model transformation sequence from the resulting target model into the given source model. By Thm. 5.2.34 we have shown that model transformations based on TGGs according to the presented approaches in Ch. 5 always ensure standard information preservation. Complete information preservation ensures that the original source model can be uniquely reconstructed from a given target model, i.e. no detail of the source model is lost during the model transformation. By Thm. 5.2.37 we have shown that complete information preservation is ensured, if the derived backward model transformation of the TGG has functional behaviour.

In order to improve the efficiency of the execution of model transformations based on TGGs we provided an on-the-fly construction for model transformations based on forward rules in Sec. 5.1.1. Furthermore, we presented the generation of forward translation rules 5.1.2, which enable the correct execution and analysis of model transformations using

highly optimized plain graph transformation engines. For this purpose, we have shown by Thm. 5.1.34 that both transformation concepts are equivalent, i.e. the formal control condition "source consistency" is automatically ensured for complete transformation sequences due to the additional translation attributes. Therefore, the results on syntactical correctness, completeness and termination are also available as shown by Cor. 5.2.6.

Further optimizations of model transformations are presented in Sec. 5.3. For this purpose, we provided powerful analysis techniques for the detection of possible conflicts between model transformation rules. Based on these results, the designer of a model transformation can detect and eliminate inconsistencies with respect to given requirements and he can reduce the conflicts by modifying the rules in order to improve efficiency. By Thm. 5.3.7 we have shown how conflicts can be resolved in a conservative way, such that the resulting model transformation relation is contained in the previous one. Moreover, we presented a static and a dynamic generation techniques for filter NACs, which filter out backtracking paths and therefore, improve efficiency. By Thm. 5.2.27 we have also shown that we can guarantee executions in polynomial time and space if all significant critical pairs for the improved rule set are strictly confluent. For the cases where backtracking cannot be avoided, we additionally provide results for analysing parallel independence (Thm. 5.3.10, Fact 5.3.11), which build the basis for partial order reduction techniques. Moreover we have shown how backtracking efforts can be reduced based on the process analysis results in Ch. 3.

9.2 Summary of Results for Visual Enterprise Modelling

In order to improve today's solutions concerning security, risk and conformance in enterprise modelling, we provided adequate automated, formal and powerful techniques for the following two typical problem areas based on the general theoretical techniques and results for graph and model transformation. The first problem area concerns the analysis and optimization of visual enterprise process models, in particular the improved support for business continuity management. The second problem area concerns the sound integration of the different existing heterogeneous model types by checking for conformance of enterprise process and enterprise service models. In particular, we provided suitable results and techniques for satisfying the requirements and solving the problems in enterprise modelling, which we described in Ch. 2.

Behaviour Analysis and Optimization of Visual Enterprise Process Models

In the area of business process modelling, we presented in Ch. 4 how the modelling process is improved concerning efficiency and quality using the new techniques for the analysis of interleaving semantics of transformation systems in Ch. 3. For this purpose, we presented

an operational semantics for the common domain specific modelling language of event driven process chains (EPCs), on which the techniques are applied. In order to additionally validate and guarantee functional and non-functional requirements of these models we provided an extended notion of EPCs called WDEPCs, which integrates additional information for data flow.

In order to improve the support for business continuity management, we presented new modelling techniques that allow for the specification of small continuity fragments, which handle single possible failures, instead of requiring the modelling of complete continuity processes. Furthermore, we presented how functional and non-functional requirements for the process can be formalized by visual graph constraints including, in particular, security constraints and external legislative requirements. Based on the standard process, the additional continuity snippets, and the formalized requirements, we provided an automated generation of complete business continuity process runs. By Thms. 4.2.7 and 4.2.10 based on Thms.3.4.12 and 3.5.3 we have shown that the generation is sound and complete with respect to the given functional and non-functional requirements.

This generation and analysis approach substantially reduces the modelling efforts and the amount of duplicates in models. Therefore, it reduces the risk of inconsistencies during the lifetime of the models, which is highly relevant due to the increasing complexity of models in several domains. Moreover, the generated process runs can be used in combination with workflow engines, such that the generated and verified continuations of running processes can be displayed in a user-friendly way and used for dynamic and fast decisions. This automated support can substantially improve the efficiency and quality of the models as well as the modelling process.

Conformance Analysis of Enterprise Process and Service Models

In order to provide automated support for the consistent integration of different domain specific enterprise modelling languages, we presented new techniques for relating heterogeneous models based on the developed techniques and results for model transformation in Ch. 5. In this context, we analysed the interdependencies between the domains of business process models and business service structure models (BSS models). BSS models specify available resources and certain structures which constrain the execution of business processes. For this reason, business process models have to conform to the given business service structure models.

The new concept of forward conformance is used to check whether models of the source domain respect the provided structures of a model in the target domain. In particular, we presented how the automated model transformation techniques in Sec. 5.1 are used for analysing and checking whether business process models conform to the given service structure model, which contains the specified available actor roles, communication chan-

nels and access rights. By Thm. 6.3.3 we have shown that the automated conformance checks based on model transformation are sound and complete using the formal results correctness and completeness (Thms. 5.2.1, 5.2.14) for model transformations based on TGGs.

Furthermore, Cor. 6.3.4 shows that the generation of business service models out of business process models automatically ensures conformance (using Thm. 5.2.1), such that the generated business service structure models can be used as basis for further refinement. This strategy can substantially improve the efficiency of modelling in the business service structure domain. By Thm. 6.4.1 we furthermore provide criteria for guaranteeing polynomial execution times for the conformance checks based on Thm. 5.2.27. Moreover, we additionally provided the notion of restricted forward conformance for the more general case, in which correspondences and dependencies between heterogeneous models concern only some structures of the models while others are not relevant. In this case, conformance checks are performed on a subset of the types of the model elements, i.e. conformance is checked for a specified view on the models.

Based on these techniques and results, models can be maintained in a decentralised and distributed way to cope with increasing complexity, which is especially important for enterprises in the financial sector, where our case study is placed.

9.3 Relevance for Model Driven Software Development

Model transformations are a key concept within model driven architecture (MDA, see [OMG01]) and model driven engineering (MDE). They are applied for transformations between different domain specific modelling languages on the same abstraction level, but also for the generation of more specific models, like program code, that can be further refined. Triple graph grammars (TGGs) have been already successfully applied in this area and the new formal concepts, results, and techniques for analysis as well as optimization further extend and improve the advantages and benefits of TGGs. Moreover, the presented techniques for behaviour analysis provide new general techniques for verifying properties of behavioural models and checking non-functional requirements as presented for business process models in Ch. 4. They additionally show good potential to be applicable to behavioural software models in an efficient way.

In the following, we discuss the main benefits of the new results for model transformations based on TGGs for model driven software development, the results of the provided case studies, further existing case studies on TGGs, and the provided tool support in this area. In particular, the presented model transformation approach ensures syntactical correctness and completeness and provides an intuitive visual notation. Moreover, the presented automated techniques for analysing and ensuring functional behaviour enable

more efficient validations of the generation of more specific models, because the amount of possible generated models is reduced to one per valid input model. Concerning model transformations on the same abstraction level, functional behaviour additionally enables the sound reflection of model changes in one model to its corresponding model in another domain.

From the application point of view, the additional techniques for the optimization of model transformations improve the efficiency of the execution and – if possible – can guarantee polynomial execution times by eliminating backtracking while completeness is still ensured. This way, model driven engineering and model based interoperability can be supported by efficient model transformations which are syntactically correct and complete and which require low maintenance efforts based on its intuitive specification and automated generation of operational rules.

Case Studies

The power and efficiency of the developed formal techniques has been shown by several case studies as summarised in the following.

The new behaviour analysis techniques concerning the interleaving semantics of general $\mathcal{M}$-adhesive transformation systems were first applied to a typed attributed graph transformation system defining the operational semantics of simplified mobile workflow oriented systems. Generation techniques were applied for deriving the complete set of equivalent system executions and for checking specific ones for equivalence. The analysed executions of the modelled system show that the new notion of permutation equivalence is the adequate notion for analysing the interleaving semantics, while the existing notions on switch equivalence are either to strict or to weak. This case study includes a benchmark, which shows the efficiency benefits of the approach compared to a direct analysis.

The above techniques were also applied for the case study on business process models, where we used the domain specific modelling language of extended event driven process chains (WDEPCs). The presented WDEPC model shows a partly simplified loan granting process, which is based on real-world processes in the financial sector. In order to analyse WDEPC models, we introduced a formal operational semantics for WDEPCs and presented, how functional and non-functional requirements can be checked and verified. In particular, we verified the four-eye principle. Furthermore, we presented how business continuity management can be supported by the automated generation and validation techniques, where we generated 126 validated standard and 252 additional continuity process runs, which can be used for dynamic decision support by a workflow engine.

The new techniques for model transformations based on TGGs were applied to the well-known case study from class diagrams to relational data base models and have shown to be of high impact. In fact, the different properties of model transformations were verified in-

cluding correctness, completeness, termination, strong functional behaviour and complete information preservation. Moreover, we presented a benchmark showing that the optimization techniques are powerful and can substantially reduce complexity. Moreover, we applied the model transformation techniques for case studies from business process models to process algebra in [BHG11] and from machine-centric business service to machine-centric IT service models in [BH10].

Several further case studies on model transformations based on graph transformation have been presented in the literature. For example, BPMN business process models are transformed into executable BPEL process models in [BE09] and UML sequence diagrams are transformed into UML state machines in [GMP10]. A model transformation from state machines to Petri nets is presented in [EEPT06], which is extended in [Gol10] based on the concept of TGGs with general application conditions in order to include more complex state machines. Furthermore, semantic correctness of the extended TGG model transformation has been verified. A model transformation from a DSL for production systems to Petri nets is presented in [EE08a], where additionally the rules of the operational semantics are transformed in order to show that the model transformation is behaviour preserving.

Finally, the new results for conformance analysis have been applied for the case study in Ch. 6. Conformance means that a model in one domain satisfies the structural requirements specified by a model in another domain. In the case study, we verified that the business process model in Ch. 4 specifying a loan granting process conforms to the given business service structure model using the notion of restricted forward conformance. In this case study, the notion of pure conformance is not applicable, because the domains of business process models and business service structure models are related only partially, i.e. the interdependencies between both models do not concern all model element types. Therefore, we restricted the process model to the relevant types and were able to show restricted forward conformance. This way, we verified that the process model including possible continuity snippets satisfies all requirements of the business service structure model concerning the specified permissions of actors for using certain communication channels and having access to resources.

Tool Support

The developed prototypical tool AGT-M (algebraic graph transformation based on Mathematica) provides a transformation and analysis environment for typed attributed graph transformation systems as well as for typed attributed triple graph transformation systems. The formal results and techniques of this thesis were applied in the different case studies using AGT-M and partly the existing tool AGG (attributed graph grammar system), which provides a powerful critical pair analysis engine. The case studies have shown that the new concepts and techniques in this thesis can be efficiently applied in a user friendly way.

As presented in detail in Ch. 7, the basis of AGT-M is the transformation engine for typed attributed graph transformation with node type inheritance and negative application conditions (NACs) for controlling the application of the transformation rules. The components of this transformation engine explicitly construct and store all graphs and morphisms according to the formal definitions, which is the basis for the behaviour analysis techniques in Ch. 3. Existing tools like AGG, however, focus only on the resulting objects of a transformation and do not provide the intermediate details for each step.

The AGT-M package for process analysis supports the analysis of interleaving semantics of graph transformation systems as presented in Ch. 3 and applied in Ch. 4 for the case study on business process analysis and business continuity management. Based on an executed transformation sequence in AGT-M the behaviour analysis engine generates the corresponding process model given by an STS. Furthermore, the engine provides the generation of the STS relations, the dependency net and the corresponding reachability graph of the dependency net. This way, the complete analysis of interleaving semantics based on the notion of permutation equivalence can be performed and was executed for the case studies.

Furthermore, AGT-M supports model transformations based on triple graph grammars (TGGs) and additionally model integrations according to [EEH08a]. In particular, AGT-M supports the specification and execution of formal triple graph transformation sequences including the control structure “source consistency”. In order to use the already existing powerful critical pair analysis engine of AGG for analysing important properties of model transformations, we have shown in Sec. 5.2.2 how triple graph transformation sequences can be equivalently performed as plain graph transformation engines. This way, we were able to analyse (strong) functional behaviour, (complete) information preservation, termination with AGG and we applied further analysis results for an optimization of the model transformations used in the case studies in Ch. 5 and 6.

To summarise, the provided results and techniques have been shown to solve main problems in current practice of enterprise modelling. In particular, they realize the main aims of the thesis as listed in Ch. 1 and especially improve the benefits of enterprise modelling concerning business continuity management and conformance analysis. Moreover, the overall general and formal foundation for the results concerning behaviour analysis and model transformation provides the basis for successful and beneficial applications in several important other domains, like model driven software development and system analysis.

9.4 Future Work

The presented techniques for behaviour analysis and model transformation have been developed with a focus on visual enterprise modelling, but they can be applied in several other

domains as well. Furthermore, the results have been verified within the general framework of $\mathcal{M}$-adhesive transformation systems, such that they can be also instantiated for applications using other high level replacement systems. In this section, we give an overview on future work concerning the general framework for graph and model transformation and visual enterprise modelling.

General Framework for Graph and Model Transformation

Different categorical frameworks for HLR systems and their interrelationship have been presented in [EGH10]. As the main result, we have shown that the framework of M-adhesive transformation systems (called vertical weak adhesive in [EGH10]) covers most of the other available frameworks. For this reason, the developed techniques in this thesis also fit in the more specific frameworks being adhesive, adhesive HLR, weak adhesive HLR and partial map adhesive transformation systems. Future work will encompass case studies for relevant instantiations in these frameworks.

In particular, we will apply the behaviour analysis techniques of Ch. 3 to the framework of reconfigurable Petri nets, which are used for modelling mobile ad hoc networks. As shown in [MGE+09], reconfigurable Petri nets form an $\mathcal{M}$-adhesive category. Together with Sarkaft Shareef [Sha10] we have shown that this category additionally has effective unions, such that we plan to apply the behaviour analysis techniques of Ch. 3 to suitable case studies.

The presented techniques for behaviour analysis and the notion of processes for $\mathcal{M}$-adhesive transformation systems in Ch. 3 have been provided concerning interleaving semantics. In a future step, the existing results will be extended towards a truly concurrent semantics, where transformation steps can be composed to concurrent steps. In some initial case studies, we experienced that a refined notion of parallel independence is necessary, in order to ensure that the modelled systems can execute these steps. Therefore, we plan to provide a new restricted notion of parallel independence based on the co-span DPO approach presented in [EHP09], which provides a more intuitive view on the relevant aspects in this case. Moreover, we plan to extend the results from transformation systems with NACs to transformation systems with general (nested) application conditions [HP09, EHL+10a]. Further improvements of efficiency could be obtained by observing the occurring symmetries in the P/T Petri net by applying symmetry reduction techniques. Additionally, the space complexity of the analysis could be reduced by unfolding the net and then representing all permutation-equivalent derivations in a more compact, partially ordered structure.

The presented techniques on model transformation in Ch. 5 are mostly defined for the general framework of $\mathcal{M}$-adhesive transformation systems already, but the results for forward translation rules are defined for the specific instantiation of typed attributed triple

graph transformation systems. As shown already in [HEGO10a], the general concept of interfaces can adequately provide an abstract representation of the used translation attributes within forward translation rules. In a next step we plan to extend the results, especially concerning functional behaviour, to the framework of $\mathcal{M}$-adhesive transformation systems based on the concept of interfaces. As shown in [GEH11], we already extended several results from triple graph grammars with NACs to systems with general (nested) application conditions.

Moreover, as presented in [BHEE10], we work on extended techniques which enable model transformations of constraints, such that model properties and requirements can be transferred to related domains. Furthermore, we aim to study semantic correctness of model transformations based on relating the operational semantics of one domain to the one of a related domain by applying the model transformation onto the rules of the operational semantics using similar ideas as presented in [EEE09] and using the concept of borrowed context [EK06]. We also plan to study additional efficient techniques for analysing and resolving conflicts between model transformation rules based on sufficient conditions for determining the type of conflict. Finally, we plan to extend the results for model integration [EEH08a] to the more general notion of model synchronization as presented in [GW09, GH09].

Visual Enterprise Modelling

While the enterprise models in the case studies of this thesis were placed in two domains, there are several further dimensions in enterprise modelling as presented in [BH10], which can be studied in more detail including further case studies. The extended enterprise modelling framework contains human-centric and machine-centric models of services, processes and rules of the Business and IT universe. The main purpose of the extended framework is to collect the distributed knowledge and requirements in order to integrate or align them in a consistent way. In this context, we plan to apply extended concepts for model integration and synchronization and to check for conformance between the new types of models.

Furthermore, the extended framework leads to new types of integration tasks, namely the integration and synchronization of several models. For this purpose, the synchronization can be performed either pairwise or in a way that synchronously considers all involved models and it is worth to study the differences concerning efficiency and applicability. Moreover, the tool support can be extended by several further techniques mentioned above and by combining a workflow engine for EPCs with the analysis and generation techniques for business processes of AGT-M.

Bibliography

[ABJ+10] Thorsten Arendt, Enrico Biermann, Stefan Jurack, Christian Krause, and Gabriele Taentzer. Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations. In Dorina C. Petriu, Nicolas Rouquette, and Øystein Haugen, editors, *Proc. Int. Conf. on Model Driven Engineering Languages and Systems (MoDELS'10)*, volume 6394 of *Lecture Notes in Computer Science*, pages 121–135. Springer, 2010.

[ABV07] Víctor Anaya, Giuseppe Berio, and Maria Jose Verdecho. Evaluating Quality of Enterprise Modelling Languages: The UEML solution. In Ricardo Jardim-Gonçalves, Jörg P. Müller, Kai Mertins, and Martin Zelm, editors, *Enterprise Interoperability II - New Challenges and Industrial Approaches, Proc. Int. Conf. on Interoperability for Enterprise Software and Applications (IESA 2007)*, pages 237–240. Springer, 2007.

[Ada10] Jochen Adamek. Konzeption und Implementierung einer Anwendungsumgebung für attributierte Graphtransformation basierend auf Mathematica. Technical Report 2009/15, TU Berlin, Fak. IV, 2010.

[AGG10] TFS-Group, TU Berlin. *AGG*, 2010. http://tfs.cs.tu-berlin.de/agg.

[Aug10] Universität Duisburg-Essen. *Augur 2*, 2010. http://www.ti.inf.uni-due.de/research/augur/index.html.

[Bal00] Paolo Baldan. *Modelling Concurrent Computations: from Contextual Petri Nets to Graph Grammars*. PhD thesis, Computer Science Department - University of Pisa, 2000.

[Bas06] Basel Committee on Banking Supervision. *Basel II: International Convergence of Capital Measurement and Capital Standards: A Revised Framework - Comprehensive Version*, 2006. http://www.bis.org/publ/bcbsca.htm.

[BBE+07] Benjamin Braatz, Christoph Brandt, Thomas Engel, Frank Hermann, and Hartmut Ehrig. An approach using formally well-founded domain languages for secure coarse-grained IT system modelling in a real-world banking scenario. In *Proc. Australasian Conf. on Information Systems (ACIS'07)*, pages 386–395, 2007.

[BCH+06] Paolo Baldan, Andrea Corradini, Tobias Heindel, Barbara König, and Pawel Sobocinski. Processes for Adhesive Rewriting Systems. In Luca Aceto and Anna Ingólfsdóttir, editors, *FoSSaCS*, volume 3921 of *Lecture Notes in Computer Science*, pages 202–216. Springer, 2006.

[BE09] Enrico Biermann and Claudia Ermel. Transforming BPMN to BPEL with EMF Tiger. In *Proc. Graph-based Tools (GraBaTs'09)*, 2009. http://is.tm.tue.nl/staff/pvgorp/events/grabats2009/.

[BEEH08] E. Biermann, K. Ehrig, C. Ermel, and J. Hurrelmann. Flexible Visualization of Automatic Simulation based on Structured Graph Transformation. In P. Bottoni and M. B. Rosson, editors, *Proc. IEEE Symposium on Visual Languages and Human-Centric Computing (VLHCC 2008)*, pages 21–28. IEEE Computer Society, 2008.

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

[BESW11] E. Biermann, C. Ermel, J. Schmidt, and A. Warning. Visual Modeling of Controlled EMF Model Transformation using Henshin. In *Proc. 4th Intern. Workshop on Graph-Based Tools (GraBaTs'10)*, 2011. To appear.

[BET08] E. Biermann, C. Ermel, and G. Taentzer. Precise semantics of EMF model transformations by graph transformation. In K. Czarnecki, editor, *Proc. ACM/IEEE 11th International Conference on Model Driven Engineering Languages and Systems (MoDELS'08)*, volume 5301 of *Lecture Notes in Computer Science*, pages 53–67. Springer, 2008.

[BH10] Christoph Brandt and Frank Hermann. How Far Can Enterprise Modeling for Banking Be Supported by Graph Transformation? In Hartmut Ehrig, Arend Rensink, Grzegorz Rozenberg, and Andy Schürr, editors, *Int. Conf. on Graph Transformation (ICGT 2010)*, volume 6372 of *Lecture Notes in Computer Science*, pages 3–26. Springer, 2010.

[BHE09a] Christoph Brandt, Frank Hermann, and Thomas Engel. Modeling and Reconfiguration of critical Business Processes for the purpose of a Business Continuity Management respecting Security, Risk and Compliance requirements at Credit Suisse using Algebraic Graph Transformation. In *Enterprise Distributed Object Computing Conference Workshops, 2009. EDOCW 2009. 13th, Proc. International Workshop on Dynamic and Declarative Business Processes (DDBP 2009)*, pages 64–71. IEEE Xplore Digital Library, 2009.

[BHE09b] Christoph Brandt, Frank Hermann, and Thomas Engel. Security and Consistency of IT and Business Models at Credit Suisse realized by Graph Constraints, Transformation and Integration using Algebraic Graph Theory. In *Proc. Int. Conf. on Ex-*

ploring Modeling Methods in Systems Analysis and Design 2009 (EMMSAD'09), volume 29 of *Lecture Notes in Business Information Processing*, pages 339–352, Heidelberg, 2009. Springer Verlag.

[BHEE10] Christoph Brandt, Frank Hermann, Hartmut Ehrig, and Thomas Engels. Enterprise Modelling using Algebraic Graph Transformation - Extended Version. Technical Report 2010/6, TU Berlin, Fak. IV, 2010.

[BHG10] Christoph Brandt, Frank Hermann, and Jan Friso Groote. Modeling and Reconfiguration of critical Business Processes for the purpose of a Business Continuity Management respecting Security, Risk and Compliance requirements at Credit Suisse using Algebraic Graph Transformation: Extended Version. Technical Report 2010/11, TU Berlin, Fak. IV, 2010.

[BHG11] Christoph Brandt, Frank Hermann, and Jan Friso Groote. Generation and Evaluation of Business Continuity Processes; Using Algebraic Graph Transformation and the mCRL2 Process Algebra. *Journal of Research and Practice in Information Technology*, 2011. To appear.

[BKS04a] Paolo Baldan, Barbara König, and Ingo Stürmer. Generating Test Cases for Code Generators by Unfolding Graph Transformation Systems. In Hartmut Ehrig, Gregor Engels, Francesco Parisi-Presicce, and Grzegorz Rozenberg, editors, *Graph Transformations, Second International Conference (ICGT'04)*, volume 3256 of *Lecture Notes in Computer Science*, pages 194–209. Springer, 2004.

[BKS04b] Benjamin Braatz, Markus Klein, and Gunnar Schröter. Semantical Integration of Object-Oriented Viewpoint Specification Techniques. In Hartmut Ehrig, Werner Damm, Jörg Desel, Martin Große-Rhode, Wolfgang Reif, Eckehard Schnieder, and Engelbert Westkämper, editors, *Integration of Software Specification Techniques for Applications in Engineering*, volume 3147 of *Lecture Notes in Computer Science*, pages 602–626. Springer, 2004.

[BN96] Peter Bernus and Laszlo Nemes. A framework to define a generic enterprise reference architecture and methodology. *Computer Integrated Manufacturing Systems*, 9(3):179 – 191, 1996.

[Boe09] Wolfgang Boehmer. Survivability and Business Continuity Management System According to BS 25999. In *Proc. Int. Conf. on Emerging Security Information, Systems and Technologies (SECURWARE '09)*, pages 142–147. IEEE Computer Society, 2009.

[Bri06] British Standards Institution (BSi). *Business continuity management. Code of practice*, 2006.

[Bri07] British Standards Institution (BSi). *Business continuity management. Specification*, 2007.

[Cha69] Alfred D. Chandler. *Strategy and Structure: Chapters in the History of the American Industrial Enterprise*. The MIT Press, 1969.

[CHS08] Andrea Corradini, Frank Hermann, and PawełSobociński. Subobject Transformation Systems. *Applied Categorical Structures*, 16(3):389–419, 2008.

[CMR96] Andrea Corradini, Ugo Montanari, and Francesca Rossi. Graph processes. *Fundamenta Informaticae*, 26(3/4):241–265, 1996.

[dLG08] Juan de Lara and Esther Guerra. Pattern-Based Model-to-Model Transformation. In Hartmut Ehrig, Reiko Heckel, Grzegorz Rozenberg, and Gabriele Taentzer, editors, *Proc. 4th Int. Conf. on Graph Transformations (ICGT 2008)*, volume 5214 of *Lecture Notes in Computer Science*, pages 426–441. Springer, 2008.

[Don09] Eugene Don. *Schaum's Outline of Mathematica*. McGraw Hill, 2009.

[DR09] Peter Dadam and Manfred Reichert. The ADEPT project: a decade of research and development for robust and flexible process support. *Computer Science - R&D*, 23(2):81–97, 2009.

[dV07] J. de Lara and H. Vangheluwe. *AToM3: A Tool for Multi-formalism and Meta-Modelling*, 2007. http://atom3.cs.mcgill.ca/.

[Ecl10] Eclipse Consortium. *Eclipse – Version 3.6*, 2010. http://www.eclipse.org.

[EE08a] H. Ehrig and C. Ermel. Semantical Correctness and Completeness of Model Transformations using Graph and Rule Transformation. In *Proc. International Conference on Graph Transformation (ICGT'08)*, volume 5214 of *Lecture Notes in Computer Science*, pages 194–210, Heidelberg, 2008. Springer.

[EE08b] C. Ermel and K. Ehrig. Visualization, Simulation and Analysis of Reconfigurable Systems. In A. Schürr, M. Nagl, and A. Zündorf, editors, *Applications of Graph Transformation with Industrial Relevance, Proceedings of the Third International AGTIVE 2007 Symposium*, volume 5088 of *Lecture Notes in Computer Science*, pages 265–281, Heidelberg, 2008. Springer.

[EEE+07] Hartmut Ehrig, Karsten Ehrig, Claudia Ermel, Frank Hermann, and Gabriele Taentzer. Information Preserving Bidirectional Model Transformations. In Matthew B. Dwyer and Antónia Lopes, editors, *Fundamental Approaches to Software Engineering*, volume 4422 of *Lecture Notes in Computer Science*, pages 72–86. Springer, 2007.

[EEE09] Hartmut Ehrig, Claudia Ermel, and Karsten Ehrig. Refactoring of Model Transformations. In Reiko Heckel and Artur Boronat, editors, *Proc. Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT'09)*, volume 18. European Association of Software Science and Technology, 2009.

[EEEP09] Hartmut Ehrig, Karsten Ehrig, Claudia Ermel, and Ulrike Prange. Consistent Integration of Models based on Views of Meta Models. *Formal Aspects of Computing*, 22 (3):327–345, 2009.

[EEH08a] H. Ehrig, K. Ehrig, and F. Hermann. From Model Transformation to Model Integration based on the Algebraic Approach to Triple Graph Grammars. *Electronic Communications of the EASST*, 10(2), 2008.

[EEH08b] H. Ehrig, K. Ehrig, and F. Hermann. From Model Transformation to Model Integration based on the Algebraic Approach to Triple Graph Grammars (Long Version). Technical Report 2008/03, TU Berlin, Fak. IV, 2008.

[EEH08c] H. Ehrig, C. Ermel, and F. Hermann. On the Relationship of Model Transformations Based on Triple and Plain Graph Grammars. In G. Karsai and G. Taentzer, editors, *Proc. Int. Workshop on Graph and Model Transformation (GraMoT'08)*. ACM, 2008.

[EEH08d] H. Ehrig, C. Ermel, and F. Hermann. On the Relationship of Model Transformations Based on Triple and Plain Graph Grammars (Long Version). Technical Report 2008/05, TU Berlin, Fak. IV, 2008.

[EEH09] H. Ehrig, C. Ermel, and F. Hermann. Transformation of Type Graphs with Inheritance for Ensuring Security in E-Government Networks. In M. Wirsing and M. Chechik, editors, *Proc. International Conference on Fundamental Aspects of Software Engineering (FASE'09)*, volume 5503 of *Lecture Notes in Computer Science*, pages 325–339. Springer, 2009.

[EEHP09a] H. Ehrig, C. Ermel, F. Hermann, and U. Prange. On-the-Fly Construction, Correctness and Completeness of Model Transformations based on Triple Graph Grammars: Long Version. Technical Report 2009/11, TU Berlin, Fak. IV, 2009.

[EEHP09b] H. Ehrig, C. Ermel, F. Hermann, and U. Prange. On-the-Fly Construction, Correctness and Completeness of Model Transformationsbased on Triple Graph Grammars. In A. Schürr and B. Selic, editors, *Proc. ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems (MODELS'09)*, volume 5795 of *Lecture Notes in Computer Science*, pages 241–255. Springer, 2009.

[EEHT97] G. Engels, H. Ehrig, R. Heckel, and G. Taentzer. A Combined Reference Model- and View-Based Approach to System Specification. *Int. Journal of Software and Knowledge Engineering*, 7(4):457–477, 1997.

[EEHT05] K. Ehrig, C. Ermel, S. Hänsgen, and G. Taentzer. Generation of Visual Editors as Eclipse Plug-Ins. In David F. Redmiles, Thomas Ellman, and Andrea Zisman, editors, *Proc. IEEE/ACM Int. Conf. on Automated Software Engineering (ASE 2005)*, pages 134–143. ACM, 2005.

[EEKR99] H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors. *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, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs in Theor. Comp. Science. Springer, 2006.

[EGH10] Hartmut Ehrig, Ulrike Golas, and Frank Hermann. Categorical Frameworks for Graph Transformation and HLR Systems based on the DPO Approach. *Bulletin of the EATCS*, 102:111–121, 2010.

[EHL+10a] Hartmut Ehrig, Annegret Habel, Leen Lambers, Fernando Orejas, and Ulrike Golas. Local Confluence for Rules with Nested Application Conditions. In Hartmut Ehrig, Arend Rensink, Grzegorz Rozenberg, and Andy Schürr, editors, *Proc. Intn. Conf. on Graph Transformation (ICGT'10)*, volume 6372 of *Lecture Notes in Computer Science*. Springer, 2010.

[EHL10b] Hartmut Ehrig, Anngegret Habel, and Leen Lambers. Parallelism and Concurrency Theorems for Rules with Nested Application Conditions. In Frank Drewes, Annegret Habel, Berthold Hoffmann, and Detlef Plump, editors, *Manipulation of Graphs, Algebras and Pictures: Essays Dedicated to Hans-Jörg Kreowski on the Occasion of His 60th Birthday*, volume 26. Electronic Communications of the EASST, 2010.

[EHP+07] H. Ehrig, K. Hoffmann, J. Padberg, U. Prange, and C. Ermel. Independence of Net Transformations and Token Firing in Reconfigurable Place/Transition Systems. In Jetty Kleijn and Alex Yakovlev, editors, *Proc. of 28th International Conference on Application and Theory of Petri Nets and Other Models of Concurrency*, volume 4546 of *Lecture Notes in Computer Science*, pages 104–123. Springer, 2007.

[EHP09] Hartmut Ehrig, Frank Hermann, and Ulrike Prange. Cospan DPO Approach: An Alternative for DPO Graph Transformations. *Bulletin of the EATCS*, pages 139–146, 2009.

[EHS09a] Hartmut Ehrig, Frank Hermann, and Christoph Sartorius. Completeness and Correctness of Model Transformations based on Triple Graph Grammars with Negative Application Conditions. *Electronic Communications of the EASST*, 18(6), 2009.

[EHS09b] Hartmut Ehrig, Frank Hermann, and Christoph Sartorius. Completeness and Correctness of Model Transformations based on Triple Graph Grammars with Negative Application Conditions (Long Version). Technical Report 2009/3, TU Berlin, 2009.

[EK06] Hartmut Ehrig and Barbara König. Deriving Bisimulation Congruences in the DPO Approach to Graph Rewriting with Borrowed Contexts. *Mathematical Structures in Computer Science*, 16(6):1133–1163, 2006.

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

[EP08] H. Ehrig and U. Prange. Formal Analysis of Model Transformations Based on Triple Graph Rules with Kernels. In H. Ehrig, R. Heckel, G. Rozenberg, and G. Taentzer, editors, *Proc. International Conference on Graph Transformation (ICGT'08)*, volume 5214 of *Lecture Notes in Computer Science*, pages 178–193, Heidelberg, 2008. Springer.

[EPS73] 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.

[Erm06] C. Ermel. *Simulation and Animation of Visual Languages based on Typed Algebraic Graph Transformation*. PhD thesis, Technische Universität Berlin, Fak. IV, Books on Demand, Norderstedt, 2006.

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

[Fed09] Federal Office for Information Security (BSI). *BSI Standard 100-4: Business Continuity Management*, 2009.

[FG98] Mark S. Fox and Michael Grüninger. Enterprise Modeling. *AI Magazine*, 19(3):109–121, 1998.

[Fra02] Ulrich Frank. Multi-perspective Enterprise Modeling (MEMO) - Conceptual Framework and Modeling Languages. In *Proc. Hawaii Int. Conf. on System Sciences (HICSS 2002)*, page 72, 2002.

[Fuj07] Software Engineering Group, University of Paderborn. *Fujaba Tool Suite*, 2007. http://wwwcs.uni-paderborn.de/cs/ag-schaefer/Lehre/PG/Fujaba/projects/tgg/index.html.

[Gd06a] E. Guerra and J. de Lara. Attributed typed triple graph transformation with inheritance in the double pushout approach. Technical Report UC3M-TR-CS-2006-00, Universidad Carlos III, Madrid, Spain, 2006.

[Gd06b] E. Guerra and J. de Lara. Model View Management with Triple Graph Grammars. In A. Corradini, H. Ehrig, U. Montanari, L. Ribeiro, and G. Rozenberg, editors, *Proc. Intern. Conf. on Graph Transformation (ICGT'06)*, volume 4178 of *Lecture Notes in Computer Science*, pages 351–366, Heidelberg, 2006. Springer.

[GEH11] Ulrike Golas, Hartmut Ehrig, and Frank Hermann. Enhancing the Expressiveness of Formal Specifications for Model Transformations by Triple Graph Grammars with Application Conditions. In *Proc. Int. Workshop on Graph Computation Models (GCM'10)*, 2011. To appear.

[GEMT00] M. Goedicke, B. Enders, T. Meyer, and G. Taentzer. ViewPoint-Oriented Software Development: Tool Support for Integrating Multiple Perspectives by Distributed Graph Transformation. In *Conf. on Tools and Algorithms for the Construction and Analysis of Systems, Berlin, Germany*, volume 1785 of *Lecture Notes in Computer Science*, pages 43 – 47. Springer, 2000.

[GH09] H. Giese and S. Hildebrandt. Efficient Model Synchronization of Large-Scale Models . Technical Report 28, Hasso Plattner Institute at the University of Potsdam, 2009.

[GHHN08] J. Grundy, J. Hosking, J. Huh, and K. Na-Liu Li. Marama: An Exclipse Metatoolset for Generating Multi-view Environments. In *Proc. Intern. Conf. on Software Engineering (ICSE'08)*, pages 819–822. ACM Press, 2008.

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

[GMP10] Roy Grønmo and Birger Møller-Pedersen. From Sequence Diagrams to State Machines by Graph Transformation. In Laurence Tratt and Martin Gogolla, editors, *Proc. Int. Conf. on Theory and Practice of Model Transformations (ICMT 2010)*, volume 6142 of *Lecture Notes in Computer Science*, pages 93–107. Springer, 2010.

[Gol10] Ulrike Golas. *Correctness and Analysis of Graph and Model Transformations*. PhD thesis, Technische Universität Berlin, Fakultät IV, 2010.

[grg06] Universität Karlsruhe. *Graph Rewrite GENerator (GrGen)*, 2006. http://www.info.uni-karlsruhe.de/software.php/id=7&lang=en.

[Gro09] Object Management Group. MDA Specifications. http://www.omg.org/mda/specs.htm, 2009.

[Gro10] *GRaphs for Object-Oriented VErification (GROOVE)*, 2010. http://groove.sourceforge.net/groove-index.html.

[GW09] H. Giese and R. Wagner. From model transformation to incremental bidirectional model synchronization. *Software and Systems Modeling*, 8(1):21–43, 2009.

[HBR09] Alena Hallerbach, Thomas Bauer, and Manfred Reichert. Guaranteeing Soundness of Configurable Process Variants in Provop. In Birgit Hofreiter and Hannes Werthner, editors, *Proc. IEEE Conf. on Commerce and Enterprise Computing, (CEC'09)*, pages 98–105. IEEE Computer Society, 2009.

[HCEK10a] Frank Hermann, Andrea Corradini, Hartmut Ehrig, and Barbara König. Efficient Analysis of Permutation Equivalence of Graph Derivations Based on Petri Nets . *Electronic Communications of the EASST*, 29(13), 2010.

[HCEK10b] Frank Hermann, Andrea Corradini, Hartmut Ehrig, and Barbara König. Efficient Process Analysis of Transformation Systems Based on Petri nets. Technical Report TR 2010/3, TU Berlin, 2010.

[HE08] Frank Hermann and Hartmut Ehrig. Process Definition using Subobject Transformation Systems. *EATCS Bulletin*, 95:153–163, 2008.

[HEGO10a] Frank Hermann, Hartmut Ehrig, U. Golas, and F. Orejas. Formal Analysis of Functional Behaviour for Model Transformations Based on Triple Graph Grammars - Extended Version. Technical Report TR 2010/8, TU Berlin, Fak. IV, 2010.

[HEGO10b] Frank Hermann, Hartmut Ehrig, Ulrike Golas, and Fernando Orejas. Efficient Analysis and Execution of Correct and Complete Model Transformations Based on Triple Graph Grammars. In *Proc. Int. Workshop on Model Driven Interoperability (MDI'10)*, pages 22–31. ACM, 2010.

[HEGO10c] Frank Hermann, Hartmut Ehrig, Ulrike Golas, and Fernando Orejas. Efficient Analysis and Execution of Correct and Complete Model Transformations Based on Triple Graph Grammars - Extended Version. Technical Report TR 2010/13, TU Berlin, Fak. IV, 2010.

[Hei10] Tobias Heindel. Hereditary pushouts reconsidered. In Hartmut Ehrig, Arend Rensink, Grzegorz Rozenberg, and Andy Schürr, editors, *Proc. Int. Conf. on Graph Transformation (ICGT 2010)*, volume 6372 of *Lecture Notes in Computer Science*, pages 250–265. Springer, 2010.

[HEOG10] Frank Hermann, Hartmut Ehrig, Fernando Orejas, and Ulrike Golas. Formal Analysis of Functional Behaviour of Model Transformations Based on Triple Graph Grammars. In Hartmut Ehrig, Arend Rensink, Grzegorz Rozenberg, and Andy Schürr, editors, *Proc. Int. Conf. on Graph Transformation (ICGT' 10)*, volume 6372 of *Lecture Notes in Computer Science*, pages 155–170. Springer, 2010.

[HEP07] K. Hoffmann, H. Ehrig, and J. Padberg. Flexible Modeling of Emergency Scenarios using Reconfigurable Systems. In *Proc. of the 10th World Conference on Integrated Design & Process Technology*, page 15, 2007. CDROM.

[Her08a] Frank Hermann. Process Construction and Analysis for Workflows Modelled by Adhesive HLR Systems with Application Conditions. In H. Ehrig, R. Heckel, G. Rozenberg, and G. Taentzer, editors, *Proc. International Conference on Graph Transformation (ICGT'08)*, volume 5214 of *Lecture Notes in Computer Science*, pages 496–498, Heidelberg, 2008. Springer.

[Her08b] Frank Hermann. Process Definition of Adhesive HLR Systems (Long Version) . Technical Report 2008/09, TU Berlin, Fak. IV, 2008.

[Her09a] Frank Hermann. Permutation Equivalence of DPO Derivations with Negative Application Conditions based on Subobject Transformation Systems. *Electronic Communications of the EASST*, 16(6), 2009.

[Her09b] Frank Hermann. Permutation Equivalence of DPO Derivations with Negative Application Conditions based on Subobject Transformation Systems: Long Version. Technical Report 2009/10, TU Berlin, Fak. IV, 2009.

[HHK10] Frank Hermann, Mathias Hülsbusch, and Barbara König. Specification and Verification of Model Transformations. *Electronic Communications of the EASST*, 30(13), 2010.

[HHT96] A. Habel, R. Heckel, and G. Taentzer. Graph Grammars with Negative Application Conditions. *Special issue of Fundamenta Informaticae*, 26(3,4):287–313, 1996.

[Hoa85] Charles Antony Richard Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[HP09] Annegret Habel and Karl-Heinz Pennemann. Correctness of high-level transformation systems relative to nested conditions. *Mathematical Structures in Computer Science*, 19(2):245–296, 2009.

[IDS10] IDS Scheer AG. *ARIS*, 2010. http://www.ids-scheer.com/.

[ISO06] International Organization for Standardization (ISO). *ISO Standard 19439:2006: Enterprise integration – Framework for enterprise modelling*, 2006.

[KHM06] Harmen Kastenberg, Frank Hermann, and Tony Modica. Towards Translating Graph Transformation Systems by Model Transformation. *Electronic Communications of the EASST*, 4(6), 2006.

[KK04] H. C. M. Kleijn and M. Koutny. Process semantics of general inhibitor nets. *Information and Computation*, 190(1):18–69, 2004.

[KK08] Barbara König and Vitali Kozioura. Augur 2—A New Version of a Tool for the Analysis of Graph Transformation Systems. In *Proc. of GT-VMT '06 (Workshop on Graph Transformation and Visual Modeling Techniques)*, volume 211 of *ENTCS*, pages 201–210. Elsevier, 2008.

[KKS07] F. Klar, A. Königs, and A. Schürr. Model Transformation in the Large. In *Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, ACM Digital Library Proceedings, pages 285–294, New York, 2007. ACM Press.

[Klj10] Olegs Kljus. Concept and Implementation of an Application Environment for Model Transformations Based on Triple Graph Grammars and Mathematica. Diplomarbeit (master's thesis), TU Berlin, Fak. IV, 2010.

[Kre86] Hans-Jörg Kreowski. Is parallelism already concurrency? Part 1: Derivations in graph grammars. In *Graph-Grammars and Their Application to Computer Science*, volume 291 of *Lecture Notes in Computer Science*, pages 343–360. Springer, 1986.

[KS06] A. Königs and A. Schürr. Tool Integration with Triple Graph Grammars - A Survey. In *Proc. SegraVis School on Foundations of Visual Modelling Techniques*, volume 148, pages 113–150. Electronic Notes in Theoretical Computer Science, Elsevier Science, 2006.

[KW07] E. Kindler and R. Wagner. Triple Graph Grammars: Concepts, Extensions, Implementations, and Application Scenarios. Technical Report TR-ri-07-284, Department of Computer Science, University of Paderborn, Germany, 2007.

[Lam09] Leen Lambers. *Certifying Rule-Based Models using Graph Transformation*. PhD thesis, Technische Universität Berlin, 2009.

[LBE+07] J. de Lara, R. Bardohl, H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. Attributed Graph Transformation with Node Type Inheritance. *Theoretical Computer Science*, 376(3):139–163, 2007.

[LBM+01] Akos Ledeczi, Arpad Bakay, Miklos Maroti, Peter Volgyesi, Greg Nordstrom, Jonathan Sprinkle, and Gabor Karsai. Composing domain-specific design environments. *IEEE Computer*, pages 44–51, November 2001.

[LEO06] L. Lambers, H. Ehrig, and F. Orejas. Conflict Detection for Graph Transformation with Negative Application Conditions. In *Proc. Third International Conference on Graph Transformation (ICGT'06)*, volume 4178 of *Lecture Notes in Computer Science*, pages 61–76. Springer, 2006.

[LEOP08] L. Lambers, H. Ehrig, F. Orejas, and U. Prange. Parallelism and Concurrency in Adhesive High-Level Replacement Systems with Negative Application Conditions. In H. Ehrig, J. Pfalzgraf, and U. Prange, editors, *Proceedings of the ACCAT workshop at ETAPS 2007*, volume 203 / 6 of *ENTCS*, pages 43–66. Elsevier, 2008.

[LRD08] Linh Thao Ly, Stefanie Rinderle, and Peter Dadam. Integration and verification of semantic constraints in adaptive process management systems. *Data Knowl. Eng.*, 64(1):3–23, 2008.

[LS04] S. Lack and P. Sobociński. Adhesive Categories. In *Proc. FOSSACS 2004*, volume 2987 of *Lecture Notes in Computer Science*, pages 273–288. Springer, 2004.

[LS05] S. Lack and P. Sobociński. Adhesive and quasiadhesive categories. *Theoretical Informatics and Applications*, 39(2):511–546, 2005.

[Mar00] Chris Marshall. *Enterprise modeling with UML: designing successful software through business analysis*. Addison-Wesley Longman Ltd., Essex, UK, UK, 2000.

[Men07] Jan Mendling. *Detection and Prediction of Errors in EPC Business Process Models*. PhD thesis, Institute of Information Systems and New Media, Vienna University of Economics and Business Administration, 2007.

[MGE+09] Tony Modica, Karsten Gabriel, Hartmut Ehrig, Kathrin Hoffmann, Sarkaft Shareef, Claudia Ermel, Ulrike Golas, Frank Hermann, and Enrico Biermann. Low- and High-Level Petri Nets with Individual Tokens. Technical Report 2009/13, Technische Universität Berlin, 2009.

[Mic10] Microsoft. Domain specific language tools, 2010. http://msdn.microsoft.com/vstudio/DSLTools/.

[Min79] Henry Mintzberg. *The structuring of organizations: A synthesis of the research*. Prentice-Hall (Englewood Cliffs, N.J.), 1979.

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

[MOF06] Object Management Group. *Meta-Object Facility (MOF), Version 2.0*, 2006. http://www.omg.org/technology/documents/formal/mof.htm.

[New42] Maxwell Herman Alexander Newman. On theories with a combinatorial definition of "equivalence". *Annals of Mathematics*, 43(2):223–243, 1942.

[NHOH10] Muhammad Naeem, Reiko Heckel, Fernando Orejas, and Frank Hermann. Incremental Service Composition Based on Partial Matching of Visual Contracts. In D. Rosenblum and G. Taentzer, editors, *Proc. Intern. Conf. on Fundamental Aspects of Software Engineering (FASE'10)*, volume 6013 of *Lecture Notes in Computer Science*, pages 123–138. Springer, 2010.

[OAS07] Organization for the Advancement of Structured Information Standards. *Web Services Business Process Execution Language Version 2.0*, 2007. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[OCL03] Object Management Group. *UML 2.0 OCL Specification*, 2003. http://www.omg.org/docs/ptc/03-10-14.pdf.

[ODtHvdA06] Chun Ouyang, Marlon Dumas, Arthur H. M. ter Hofstede, and Wil M. P. van der Aalst. From BPMN process models to BPEL web services. In *IEEE International Conference on Web Services (ICWS 2006), 18-22 September 2006, Chicago, Illinois, USA*, pages 285–292. IEEE Computer Society, 2006.

[Off97] United States General Accounting Office. Business Process Reengineering Assessment Guide, 1997.

[OGdLE09] Fernando Orejas, Esther Guerra, Juan de Lara, and Hartmut Ehrig. Correctness, Completeness and Termination of Pattern-Based Model-to-Model Transformation.

In Alexander Kurz, Marina Lenisa, and Andrzej Tarlecki, editors, *Int. Conf. on Algebra and Coalgebra in Computer Science (CALCO'09)*, volume 5728 of *Lecture Notes in Computer Science*, pages 383–397. Springer, 2009.

[OMG01] Object Management Group. *Model Driven Architecture (MDA)*, 2001. http://www.omg.org/cgi-bin/doc?ormsc/01-07-01.pdf.

[OMG07] Object Management Group. *Unified Modeling Language: Superstructure – Version 2.1.1*, 2007. formal/07-02-05, http://www.omg.org/technology/documents/formal/uml.htm.

[OMG09] Object Management Group. *Business Process Model and Notation (BPMN) Version 1.2*, 2009. http://www.omg.org/spec/BPMN/1.2.

[OMG10] Object Management Group. *Unified Modeling Language: Superstructure – Version 2.3*, 2010. http://www.omg.org/technology/documents/modeling_spec_catalog.htm.

[PEH07] J. Padberg, H. Ehrig, and K. Hoffmann. Formal Modeling and Analysis of flexible Processes in Mobile Ad-Hoc Networks. *Bulletin of the EATCS*, 91:128–132, 2007.

[Pen09] Karl-Heinz Pennemann. *Development of Correct Graph Transformation Systems*. PhD thesis, Department of Computing Science, University of Oldenburg, Oldenburg, 2009. http://oops.uni-oldenburg.de/volltexte/2009/948/Electronic Dissertation.

[Plu93] Detlef Plump. Hypergraph Rewriting: Critical Pairs and Undecidability of Confluence. In *Term Graph Rewriting: Theory and Practice*, pages 201–213. John Wiley, 1993.

[Plu05] Detlef Plump. Confluence of Graph Transformation Revisited. In *Processes, Terms and Cycles: Steps on the Road to Infinity: Essays Dedicated to Jan Willem Klop on the Occasion of His 60th Birthday*, volume 3838 of *Lecture Notes in Computer Science*, pages 280–308. Springer, 2005.

[PS07] Viara Popova and Alexei Sharpanskykh. A Formal Framework for Modeling and Analysis of Organizations. In Jolita Ralyté, Sjaak Brinkkemper, and Brian Henderson-Sellers, editors, *Proc. IFIP WG 8.1 Working Conf. on Situational Method Engineering: Fundamentals and Experiences 2007*, volume 244 of *IFIP*, pages 343–358. Springer, 2007.

[QVT08] Object Management Group. *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. Version 1.0 formal/08-04-03. http://www.omg.org/spec/QVT/1.0/*, 2008.

[RD98] Manfred Reichert and Peter Dadam. ADEPT flex -Supporting Dynamic Changes of Workflows Without Losing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.

[RE96] G. Rozenberg and J. Engelfriet. Elementary Net Systems. In W. Reisig and G. Rozenberg, editors, *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 12–121. Springer, 1996.

[Rei85] Wolfgang Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1985.

[Ren04a] A. Rensink. Representing First-Order Logic Using Graphs. In H. Ehrig, G. Engels, F. Parisi-Presicce, and G. Rozenberg, editors, *International Conference on Graph Transformations (ICGT)*, volume 3256 of *Lecture Notes in Computer Science*, pages 319–335, Berlin, 2004. Springer Verlag.

[Ren04b] Arend Rensink. The GROOVE Simulator: A Tool for State Space Generation. In J. Pfalz, M. Nagl, and B. Böhlen, editors, *Applications of Graph Transformations with Industrial Relevance (AGTIVE)*, volume 3062 of *Lecture Notes in Computer Science*, pages 479–485. Springer-Verlag, 2004.

[RMRW08] Stefanie Rinderle-Ma, Manfred Reichert, and Barbara Weber. On the Formal Semantics of Change Patterns in Process-Aware Information Systems. In Qing Li, Stefano Spaccapietra, Eric S. K. Yu, and Antoni Olivé, editors, *Proc. Int. Conf. on Conceptual Modeling (ER'08)*, volume 5231 of *Lecture Notes in Computer Science*, pages 279–293. Springer, 2008.

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

[SAB98] Monique Snoeck, Rakesh Agarwal, and Chiranjit Basu. Enterprise Modelling. In Serge Demeyer and Jan Bosch, editors, *Proc. Workshops at the Europ. Conf. on Object-Oriented Programming (ECOOP 1998)*, volume 1543 of *Lecture Notes in Computer Science*, pages 222–227. Springer, 1998.

[Sch86] David A. Schmidt. *Denotational semantics: a methodology for language development*. William C. Brown Publishers, Dubuque, IA, USA, 1986.

[Sch94] A. Schürr. Specification of Graph Translators with Triple Graph Grammars. In G. Tinhofer, editor, *WG94 20th Int. Workshop on Graph-Theoretic Concepts in Computer Science*, volume 903 of *Lecture Notes in Computer Science*, pages 151–163, Heidelberg, 1994. Springer Verlag.

[Sch01] August-Wilhelm Scheer, editor. *ARIS-Modellierungs-Methoden, Metamodelle, Anwendungen*. Springer, 2001.

[SE09] Michael Stieghahn and Thomas Engel. Law-aware access control for international financial environments. In Yannis Kotidis, Pedro José Marrón, Le Gruenwald, and Demetrios Zeinalipour-Yazti, editors, *International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE'09)*, pages 33–40. ACM, 2009.

[Sha10] Sarkaft Shareef. Formal Modelling and Analysis of Reconfigurable Object Nets Based on the RON Editor. Diplomarbeit (master's thesis), TU Berlin, Fak. IV, 2010.

[SK08] Andy Schürr and Felix Klar. 15 years of triple graph grammars. In *Proc. Int. Conf. on Graph Transformation (ICGT 2008)*, pages 411–425, 2008.

[SN00] August-Wilhelm Scheer and Markus Nüttgens. ARIS Architecture and Reference Models for Business Process Management. In Wil M. P. van der Aalst, Jörg Desel, and Andreas Oberweis, editors, *Business Process Management*, volume 1806 of *Lecture Notes in Computer Science*, pages 376–389. Springer, 2000.

[Ste10] Perdita Stevens. Bidirectional model transformations in QVT: semantic issues and open questions. *Software and System Modeling*, 9(1):7–20, 2010.

[Sto77] Joseph E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*. MIT Press, Cambridge, MA, USA, 1977.

[Tae06] G. Taentzer. Towards Generating Domain-Specific Model Editors with Complex Editing Commands. In *Proc. International Workshop Eclipse Technology eXchange (eTX), Satellite Event of European Conf. on Object-Oriented Programming (ECOOP)*, 2006.

[TBB+08] G. Taentzer, E. Biermann, D. Bisztray, B. Bohnet, I. Boneva, A. Boronat, L. Geiger, R. Geiß, A. Horvath, O. Kniemeyer, T. Mens, B. Ness, D. Plump, and T. Vajk. Generation of Sierpinski Triangles: A Case Study for Graph Transformation Tools. In A. Schürr, M. Nagl, and A. Zündorf, editors, *Proc. Int. Symp. on Applications of Graph Transformation with Industrial Relevance (AGTIVE 2007)*, volume 5088 of *Lecture Notes in Computer Science*, pages 514 – 539, Heidelberg, 2008. Springer.

[TCSE08] G. Taentzer, A. Crema, R. Schmutzler, and C. Ermel. Generating Domain-Specific Model Editors with Complex Editing Commands. In A. Schürr, M. Nagl, and A. Zündorf, editors, *Proc. Int. Symp. on Applications of Graph Transformation with Industrial Relevance (AGTIVE 2007)*, volume 5088 of *Lecture Notes in Computer Science*, pages 98–103, Heidelberg, 2008. Springer.

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

[Tig10] Tiger Project Team, Technische Universität Berlin. *Tiger: Generating Visual Environments in Eclipse*, 2010. http://www.tfs.cs.tu-berlin.de/tigerprj.

[TK09] Juha-Pekka Tolvanen and Steven Kelly. MetaEdit+: defining and using integrated domain-specific modeling languages. In Shail Arora and Gary T. Leavens, editors,

Companion to the ACM SIGPLAN Conf. on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2009), pages 819–820. ACM, 2009.

[Tol07] Juha-Pekka Tolvanen. MetaEdit+: Domain-Specific Modeling and Product Generation Environment. In *Int. Conf. on Software Product Lines (SPLC 2007)*, pages 145–146. Kindai Kagaku Sha Co. Ltd., Tokyo, Japan, 2007.

[TU 10] TU Darmstadt. *MOFLON 1.4*, 2010. http://www.moflon.org.

[TvdAS09] Nikola Trcka, Wil M. P. van der Aalst, and Natalia Sidorova. Data-Flow Anti-patterns: Discovering Data-Flow Errors in Workflows. In Pascal van Eck, Jaap Gordijn, and Roel Wieringa, editors, *Int. Conf. on Advanced Information Systems Engineering (CAiSE 2009)*, volume 5565 of *Lecture Notes in Computer Science*, pages 425–439. Springer, 2009.

[Uni02] United States Code. Sarbanes-Oxley Act of 2002, PL 107-204, 116 Stat 745. Codified in Sections 11, 15, 18, 28, and 29 USC, 2002.

[vdADG+08] Wil M. P. van der Aalst, Marlon Dumas, Florian Gottschalk, Arthur H. M. ter Hofstede, Marcello La Rosa, and Jan Mendling. Correctness-Preserving Configuration of Business Process Models. In José Luiz Fiadeiro and Paola Inverardi, editors, *Int. Conf. on Fundamental Approaches to Software Engineering (FASE 2008)*, volume 4961 of *Lecture Notes in Computer Science*, pages 46–61. Springer, 2008.

[vdAHKB03] W. M. P. van der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. *Distrib. Parallel Databases*, 14(1):5–51, 2003.

[Ver02] F. Vernadat. UEML: Towards a Unified Enterprise Modelling Language. *International Journal of Production Research*, 40(17):4309 – 4321, 2002.

[VIS09] Microsoft. *Microsoft Visio*, 2009. http://office.microsoft.com/en-us/visio/.

[Win93] Glynn Winskel. *The formal semantics of programming languages: an introduction*. MIT Press, Cambridge, MA, USA, 1993.

[Wol10] Wolfram Research. *Wolfram Mathematica 7.0*, 2010. http://www.wolfram.com.

[ZSLZ08] A. Zalewski, P. Sztandera, M. Ludzia, and M. Zalewski. Modeling and Analyzing Disaster Recovery Plans as Business Processes. In *SAFECOMP '08: Proc. Int. Conf. on Computer Safety, Reliability, and Security*, volume 5219 of *Lecture Notes in Computer Science*, pages 113–125. Springer, 2008.

Index