Security and Consistency of IT and Business Models at Credit Suisse realized by Graph Constraints, Transformation and Integration using Algebraic Graph Theory (Long Version)

=== Draft ===

Christoph Brandt[1], Frank Hermann[2],
Hartmut Ehrig[2], Thomas Engel[1],
Jochen Adamek[2] and Hanna Schölzel[2]

[1] Université du Luxembourg, SECAN-Lab, Campus Kirchberg,
6, rue Richard Coudenhove-Kalergi, L-1359 Luxembourg-Kirchberg, EU
{christoph.brandt,thomas.engel}(at)uni.lu,
WWW home page: http://wwwen.uni.lu/recherche/fstc

[2] Technische Universität Berlin,
Fakultät IV, Theoretische Informatik und formale Spezifikation,
Sekr. FR 6-1, Franklinstr. 28/29, 10587 Berlin, EU
{frank,ehrig}(at)cs.tu-berlin.de,
WWW home page: http://www.tfs.tu-berlin.de

Bericht-Nr. 2009/XX
ISSN 1436-9915

Abstract

An analysis of today's situation at Credit Suisse has shown severe problems, because it is based on current best practices and ad-hoc modelling techniques to handle important aspects of security, risk and compliance. Based on this analysis we propose in this paper a new enterprise model which allows the construction, integration, transformation and evaluation of different organizational models in a big decentralized organization like Credit Suisse. The main idea of the new model framework is to provide small decentralized models and intra-model evaluation techniques to handle services, processes and rules separately for the business and IT universe on one hand and for human-centric and machine-centric concepts on the other hand. Furthermore, the new framework provides inter-modelling techniques based on algebraic graph transformation to establish the connection between different kinds of models and to allow integration of the decentralized models. In order to check for security, risk and compliance in a suitable way, our models and techniques are based on different kinds of formal methods. In this paper, we show that algebraic graph transformation techniques are useful not only for intra-modelling – using graph grammars for visual languages and graph constraints for requirements – but also for inter-modelling – using triple graph grammars for model transformation and integration.

Altogether, we present the overall idea of our new model framework and show how to solve specific problems concerning intra- and inter-modelling as first steps. This should give evidence that our framework can also handle important other requirements for enterprise modelling in a big decentralized organization like Credit Suisse.

Keywords: real-world banking scenario, service and process architecture, security policies, business and IT universe, model integration and transformation, graph constraints, algebraic graph transformation

1 Introduction

The problem addressed in this paper is about how to construct, integrate, transform and evaluate new organizational models in order to improve today's enterprise models at Credit Suisse and similar organizations based on a research project supported by Credit Suisse. The new models are going to be constructed to represent relevant aspects of the real-world organization that are needed to check for security, risk and compliance issues the bank has to handle because of national and international laws, financial regulations and internal rules.

After an analysis of today's situation at Credit Suisse and a detailed requirement analysis, this paper presents a potential solution for enterprise modelling using algebraic graph transformation. This new modelling framework includes services, processes and rules , business and an IT views and a distinction between human-centric and machine-centric models. The idea is to have not one single bulky enterprise model but multiple lean models that can be integrated leading to a holistic organizational view of the enterprise. The integrated enterprise model can be used for evaluation, simulation and automation of business processes and can be managed independently in order to match the requirements of a decentralized organization. Algebraic graph transformation is used as a formalism to construct, integrate, transform and evaluate the various models. The foundations of algebraic graph transformation date back to the 1970s [1] and the framework is continuously extended and adapted for an increasing area of applications, especially in computer science. A wide range of results for the analysis of rule based systems is available concerning correctness, confluence, concurrency and distributed computing [2, 3, 4]. The formal basis of algebraic graph transformation [5] enables formal proofs about the qualities of operations and model transformations that may impact security, risk and compliance issues. This has the potential to be of enormous practical relevance in the Credit Suisse scenario.

Therefore the scope of this study is to show how lean organizational models can be constructed and maintained in a controlled way, and how the interplay of models can be organized by exploiting techniques based on algebraic graph transformation respecting real-world requirements. This result is assumed to create the precondition for going much deeper into the analysis of security, risk and

compliance issues later on. In a first step, a security example is exemplarily discussed by applying the introduced formalisms.

Cost reduction and quality assurance as well as automation have been defined by Credit Suisse as driving business interests. Hence, the solution is not only required to lead to scientifically sound results but is also required to realize these business interests. Because of the generality of these requirements, our proposed new enterprise model can also be applied to other organizations having similar requirements.

The novelty presented in this paper lies in the combination of human-centric and machine-centric models and in using algebraic graph transformation as the formal modelling technique to construct, integrate, transform and evaluate these models. At the same time, the proposed solution has the potential to meet organizational side-constraints at Credit Suisse like decentralization and other business interests. In contrast to today's practice at Credit Suisse which is about integrating tools to address very specific issues, the proposed solution suggests to start with the question of how a holistic organizational model should be build, managed and evaluated that can answer security, risk and compliance questions. This enables the automation, monitoring and analysis of the organizational workflows and helps to facilitate a corresponding IT implementation. The proposed approach frees the enterprise model from IT implementation aspects and helps to keep the organizational model consistent and focussed to answer questions about security, risk and compliance.

Today's Credit Suisse's product-driven approach to model organizational structures and to check for security, risk and compliance requirements does not result in a holistic organizational model that can be used to answer such questions.

In contrast to any product-driven solution the proposed approach promises a better scalability and extensibility as well as lower long-term costs of ownership because it considers organizational scheme and instance models to be objects of their own and independent of implementation issues and tools. This is very different from today's practice where tools own certain data about the organization that cannot easily be used without them. Since people at Credit Suisse cannot be expected to be technical experts in formal methods it is important to point out that algebraic graph transformation is not only a formal but also a visual specification technique, which allows intuitive understanding. In detail, the construction, transformation, integration and evaluation of organizational scheme and instance models can be realized by algebraic graph transformation techniques that can directly work on the abstract syntax of these models.

This promises to enable the full control of organizational predicates that are of high practical relevance. The product-driven approach that focuses on the technical integration of tools and discards the formal integration of methods is not able to do that.

The paper is organized as follows: In Sec. 2, enterprise modelling in the sense of our new modelling framework is introduced based on the real-world requirements coming out of the Credit Suisse scenario. In detail, diagrams representing IT and business services and processes as well as samples of a domain language for organizational security requirements are shown as human-centric models. In accordance with these models abstract behaviour types and reo-connectors as well as suitable other formal techniques are proposed as machine-centric models. In sections 3 and 4 intra-model and inter-model techniques are introduced based on algebraic graph transformation techniques. They are used to construct, integrate, transform and evaluate the models of this framework. In more detail, the intra model techniques encompass the construction of models, their correction according to well-formedness rules and a validation of a subset of functional and non-functional requirements. Regarding inter-model techniques triple-graph grammars [6, 7] are explained and then used for model transformation as well as model integration. Triple-graph grammars can even help to propagate requirements that are formalized as graph constraints. In Sec. 5, we study related work and in Sec. 6, we conclude with a summary of main achievements and aspects of future work.

2 Enterprise Models and Enterprise Modelling

The first part of this section presents how Credit Suisse addresses security, risk and compliance issues today by the help of best practices. This leads to highly focussed and partial enterprise models that are mostly build using ad-hoc modelling techniques. Thereafter, we propose our new methodological approach that starts with lean and focussed enterprise models which in a later stage should be soundly integrated towards a holistic organizational view, so that security, risk and compliance issues can be answered based on an integrated and well focused organizational model of the enterprise.

2.1 Today's Situation

Today's situation at Credit Suisse is essentially top-down. It is top-down because based on given security, risk or compliance requirements specific data are collected and specific organizational operations are put into place. In the best case this results in highly specialized and partial enterprise models and in an ad-hoc modelling technique based on best-practices that are used to build those models. Therefore, security, risk and compliance departments work in a non-integrated and uncoordinated fashion. In a lot of cases controls are established after the organizational practice is already in place. So, controls are mostly realized in an end-of-pipe way. Because controls are evaluated using best-practices like checklists this often results in partial insights only. Security, risk and compliance requirements are treated on a demand basis driven by certain stakeholders who only focus on their interests. Because this results in redundant tasks and conceptual mismatches of results as well as neglected reuse potentials in the area of enterprise modelling, one could think of using a performance indicator that measures blind and misdirected outputs to characterize the maturity of the current practice at Credit Suisse. However, specific controlling data related to the potential solution were not available at the time of this writing. The following table in Fig. 1 summarizes the situation.

	Today
Approach	Best Practices
Focus	Interest
Control	End-of-pipe
Judgment	Checklists
Coverage	Partial

Figure 1: Security, Risk and Compliance – Today

Today's enterprise modelling at Credit Suisse can either be looked at from its perspective of modelling business processes in business departments or from the point of view of running technical components in IT departments. In addition to that, business and IT requirements are documented. At the bank, both perspectives exist and both have their own history, document types and people. These department-oriented views are not synchronized, not integrated, not conflict-free and sometimes over- or under-specified. The situation can further be characterized as document-oriented, not model-centric. In the documents, different types of models or fragments of models can be found – the following table in Fig. 2 lists some of them.

	Components	Processes	Requirements
Business Department	–	UML	BO
IT Department	MS Visio	–	MS Excel MS Word

Figure 2: Today's Enterprise Modelling at Credit Suisse

So, landscapes of IT components are documented using Microsoft Visio, business processes are

documented using UML [8], business requirements are documented using business object models (BO) [9] or just natural or standardized business languages. The fact that business services and IT processes are not documented in the scenario shows a mismatch in the involved departments' understanding of the common universe.

Therefore, the situation at Credit Suisse is unsatisfactory. Members of Credit Suisse' staff confirmed that best-practices do not scale well in a big organization and they do not fully cover all possible organizational states. So, there are quality and costs problems when implementing best-practices and processing the obtained results. The interest-driven data collection leads to partial models that are not synchronized and not soundly integrated. So, for example, the modelling of business continuity processes is not properly integrating security requirements. The end-of-pipe control defined by best-practices increases the cycle times for evaluations of organizational settings. So, a contemporary reaction is getting difficult. Organizational attributes that are part of best-practices' checklists are not formally grounded in well-defined enterprise models. So, there is no clear control about the derivation and deduction of security, risk and compliance predicates. Often checklists are applied to specific cases only. So, concrete statements about security, risk and compliance are exclusively valid for very specific assumptions. Therefore, they may not even be comparable among themselves.

The proposed new solution that is presented in the following addresses these issues by the help of formal methods. However, because formal methods are often considered to be difficult to understand special accentuation is put on usability and applicability as well as visual techniques and suitable implementation aspects. The aim is to replace the semi-formal and non-integrated organizational models and methods used today by a generic enterprise model build by the help of declarative and well-defined modelling techniques. The formal basis of such a model and corresponding methods will support the modelling process and the evaluation of models resulting in lower costs and better quality of results. This is very likely to improve the organizational competitiveness by the help of a better organizational control and decision support as well as automation.

At the same time organizational side constraints remain and must be respected. Today, organizational knowledge is available only partial, it is available in a non-integrated and distributed way and is often inconsistent and incomplete.

2.2 Tomorrow's Situation

Tomorrow's situation is likely to be different. Business people at Credit Suisse requested to work with models that are build using diagram- and text-oriented domain languages in an integrated way. They further requested support for fuzzy, incomplete and inconsistent models as well as for partial redundant models that have been created and entered by different persons. Because of the dynamic nature of their business environment they requested support for evolving domain languages in terms of their syntax and semantics and support to merge models from different people as well as an appropriate versioning. Despite the semi-formal nature of those models they asked for model-checking-like capabilities. Their idea of a modelling process is the one of using *clickable* mathematics which means constructing models by the help of intuitive and declarative click operations on a screen that result transparently in models that can be evaluated by the help of formal methods.

The first shift compared with today's situation is to start building fragments of an envisioned enterprise model in contrast of checking specific cases regarding concrete security, risk and compliance requirements. Therefore, our new approach starts bottom-up. Once, an appropriate enterprise model is available security, risk and compliance predicates can be defined that make reference to such a model. Therefore, their semantics will be well-defined. Corresponding security, risk and compliance requirements can then be build on top of such predicates in a constructive manner.

Further, by starting with an enterprise model the focus shifts from specific interests of certain stakeholders towards a more holistic understanding of the organization. Once, such a holistic enterprise model is available it can be model checked regarding security, risk and compliance

which causes a shift from evaluating the modelled business reality of Credit Suisse in an ex-post fashion to evaluating an enterprise model under development in an ex-ante way. So, there is a switch from an end-of-pipe control to a begin-of-pipe control. Assumed, that an enterprise model is available it would be possible to run proves, simulations and tests to check for security, risk and compliance requirements covering all states of the model. The table in Fig. 3 summarizes some characteristics of tomorrow's situation.

	Today	Tomorrow
Approach	Best Practices	formal Models, Methods
Focus	Interest	Organization
	top-down	bottom-up
Control	End-of-pipe	Begin-of-pipe
Judgment	Checklists	Prove, Simulation, Test
Coverage	Partial	Complete

Figure 3: Security, Risk and Compliance – Today and Tomorrow

Based on future requirements there will be a switch from the use of UML models and informal MS Visio diagrams at Credit Suisse towards domain specific modelling languages. In addition to that, the process oriented view of business departments will be enhanced by an understanding of business services. The same applies to the IT departments in a symmetric way. Here, the focus on IT services will be enlarged by looking at IT processes as well. At the end, Business and IT departments can work with the same type of service, process and rules models, which will facilitate any kind of automatic alignment between both worlds. Given that life cycles of models in the Business and IT universe are different and that there is no clear top-down or bottom-up dependency but a mutual relationship between both worlds, implementation driven modelling processes wont be the driving force any longer. They will be substituted by specification oriented modelling processes which are going to be orthogonal to any implementation activity. By aligning domain specific models with formal models the semantics of domain models can be explained. In detail, this relationship between so called human-centric models or domain models and machine-centric models or formal models can be realized by model integration. In the given case of this study at Credit Suisse, abstract behaviour types and reo connectors are proposed to explain the semantics of services and service landscape models, the mCRL2 process algebra [10] can be used to define the semantics of business processes that appear as event driven process chains. Finally, first-order logic can be used to explain the semantics of business rules. Such a framework of formal methods has the potential to reduce today's under-specification of enterprise models' semantics that causes significant integration problems. Organizational security, risk and compliance predicates can then be soundly anchored in this formal framework.

	Service Landscapes	Process Landscapes	Rules and Principles
Business Universe	ABT & Reo	PA & ML	FOL
IT Universe	ABT & Reo	PA & ML	FOL

Figure 4: Tomorrow's Enterprise Engineering at Credit Suisse – Machine-Centric View

The machine-oriented slice of the scenario is presented by the table in Fig. 4, where ABT & Reo means abstract behaviour types and Reo connectors [11], PA & ML means a process algebra encompassing a modal logic [10] and FOL denotes first order logic. More details are given in Sections 2.5 and 2.6.

2.3 The New Model Framework in a Nutshell

The main idea of the new model framework is to reduce the overall complexity of one big organizational model by splitting it in three dimensions as shown in Fig. 5. The first dimension includes services, processes and rules, the second dimension the business and IT universe and the third dimension human-centric and machine-centric concepts as discussed in the previous sections. Altogether this leads to 12 different types of models, which can be specified, interrelated and integrated by different techniques.

From a formal point of view the model framework in Fig. 5 shows different coordinates (X, Y, Z) and each of them usually contains several models for one overall enterprise model and these models change over time. For instance the coordinate (S, B, M) represents all service models for the business universe using a machine centric modelling language - in our scenario ABT-Reo diagrams. We denote the set of the models in one coordinate by $\boldsymbol{\mathcal{M}}_{X,Y,Z}$ and refer to a specific model by the notation $M_{X,Y,Z}$ for a model $M_{X,Y,Z} \in \boldsymbol{\mathcal{M}}_{X,Y,Z}$.

As pointed out already, the splitting into three dimensions leads to several lean and focussed models for specific purposes, which reduces the overall complexity of one big organizational model. But the new model framework includes also the interrelations between corresponding models in each of the three dimensions. The inter-dimensional interplay for each dimension will be explained below.

Last but not least our new enterprise modelling is based on algebraic graph transformation techniques, which support construction, integration and transformation of organizational models in our new framework as discussed in the next subsection.

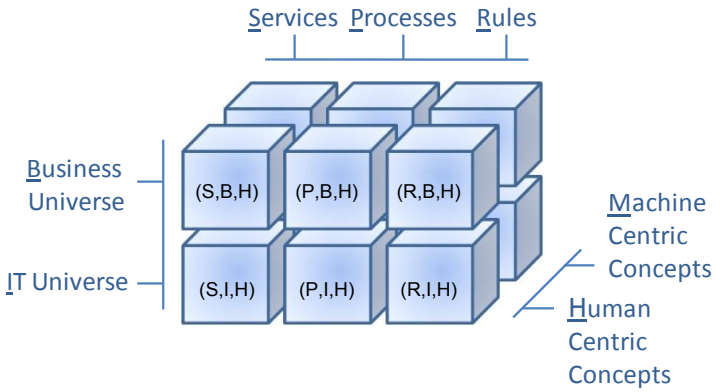


Figure 5: Model Framework

The inter-dimensional interplay in the framework in Fig. 5 between services, processes and rules can be explained by the paradigm of a street map. The service landscape is considered to be the street map on which processes run like bus lines. Rules will govern concrete decisions like it is the case for road traffic regulations.

The inter-dimensional interplay between Business and IT models is the one of a mutual supportive alignment. Parts of the Business model can be automated by the help of a corresponding IT model. However, it is still possible to assume that there are parts of the Business model without IT support. The same is true for an IT model. Parts of an IT model may automate a Business model. However, the IT may have their own specific services, processes and rules for which there is no correspondence in the business model. Therefore, neither a top-down nor a bottom-up relationship is appropriate here. It is more a relationship between equals.

The inter-dimensional interplay in the framework between human-centric and machine-centric models is about to join the world of humans with the world of machines. Humans like to have the possibility to enter incomplete, inconsistent, redundant and evolving models. In addition to that, they even like the definition of the used domain languages to be open. Such requirements are usually incompatible with implemented formal methods. However, having machine-oriented

models that are build on implemented formal methods like the ones just introduced the semantics of human-centric models can be smoothly explained by a model integration between human-centric and machine-centric models. Further, model checking techniques can be applied in an encapsulated way using machine-centric models to evaluate security, risk and compliance requirements in an automated fashion. This is something people at Credit Suisse have strongly requested. They like to use the power of formal methods without touching them. A detailed motivation is given in [12]. The special issue here is that such an integration is open. Integration rules can be changed, deleted and added. Therefore, the semantics of human-centric models can evolve. In addition to that such a loose coupling between human- and machine-centric models enables to have syntactic elements in a domain language for which there is no semantics available, but which can be added later. And assuming that a given domain model is going to be modified an aligned machine-centric model can drive completion rules during the modelling process. Therefore, we do have no clear top-down or bottom-up relationship between human- and machine-centric models.

The inter-dimensional model interplay in the framework in Fig. 5 opens an orthogonal problem and solution space. It is a problem space because the current situation of an organization can be documented and evaluated ex-post. It is a solution space because possible future organizations can be defined by it and evaluated ex-ante. Having the current and future organization available a transition path can be defined. So, not only versions of service, process and rule models but also versions of whole organizational models can be imagined. Having a history of organizational models and performance data available the practical impact of business reengineering projects can be much better controlled.

2.4 Using Algebraic Graph Transformation

Algebraic graph transformation techniques are able to support a wide range of the requirements discussed in the previous Subsections 2.2 and 2.3. So, model integration can be used to realize the alignment between Business and IT models [13, 14] using triple graph grammar techniques in Sec. 4. Because model integration works both ways round-trip-engineering can be supported in principle as it can be realized by intra-model integration and transformation techniques shown in later sections. This ideally supports the decentralized organization of Credit Suisse where models are created at different locations asynchronously. Algebraic graph transformation can be used to connect domain models or human-centric models with formal models or machine-centric models to explain the semantics of domain models while keeping this relationship open for future changes. As a consequence, machine-centric models allow, for example, fully automated evaluations of organizational policies of business processes by keeping the business process model human-centric and aligned with the machine-centric counterparts using inter-model integration techniques. Algebraic graph transformation can also support consistency checks between service, process and rules models by the help of model integration. These means to integrate models enables to keep the models independent, lean, focused and manageable and they further enable aggregated views that can be model checked. In addition to that, integrated models will facilitate the communication between different departments. Simulation models do not need to be given explicitly. They can be created by transformation rules using service, process and rule models as their input.

Algebraic graph transformation - as shown in Sec. 3 below - enables to specify graph transformation rules in a declarative way which makes this formal technique very usable. Further, the underlying algorithms can be fully implemented. At the same time, type graphs and graph transformation rules can change. Therefore, the implementation can be kept static while being able to cope with changing type graphs and changing graph transformation rules. So, transformation rules can be executed by a generic implementation of algebraic graph transformation. We will show this in detail based on a small showcase that has been implemented in Mathematica [15, 6]. Compared with alternative solutions this formal technique is built on a body of theory that allows formal proofs. Therefore, guarantees can be given which is of high relevance when discussing security, risk and compliance predicates.

In detail, models are considered as objects of their own. Small models can be integrated to

one holistic model. Decentralized modelling is possible. Enforcement of security requirements by the help of graph constraints is available. Algebraic graph transformation has a good potential to be compatible with the concept of pluggability and the notion of ontologies. Instance- and class-modelling is possible likewise. Models can be modified in a controlled way. Models can be build independently and decentralized. Transformation rules are declarative and therefore very easy to understand by people who are not experts in formal methods. Language artefacts of different types like diagram-artefacts and text-artefacts can be glued together based on their abstract syntax using the same technique. So, domain languages can be used equally easy as diagram languages. The separation into human-centric models and machine-centric models helps people to administrate incomplete or inconsistent models without being forced to comply to well-formedness of models based on formal methods. By using catalogues of model elements and assemblies a lego-like modelling system could be provided. The same applies to transformation rules or sets of transformation rules. The technique of algebraic graph transformation is able to cope with such open catalogues.

The subjectivity of entered models can be handled by intersecting service, process or rules models created by different people to derive at a common denominator that is shared by all modelling parties. Linguistic ambiguities can be addressed by integration rules that are mapping human-centric models towards machine-centric models and using redundant elements of a human-centric model for this mapping to reduce or resolve ambiguity issues. Further, the modelling process as it is supported by algebraic graph transformation is agile. It can therefore support refinements and extensions at any level of detail and abstraction in a scalable and declarative way. Finally, rules sets of transformation rules can be modified. New rules can be added, old rules can be deprecated. So, adaptive modelling can easily be supported. And because the applied technique of algebraic graph transformation address the abstract syntax of models, it can in principle handle any type of model visual or textual. It is even possible to combine semi-structured models as in the case of natural text with structured models like the ones used for business rules.

Finally, model-based interoperability is defined in the context of this study in Sec. 4 as the ability to execute sound model integrations and to check for inconsistencies during the integration as well as to execute model transformations using inter-model techniques. It further encompasses the ability to propagate model invariants – defined by graph constraints – and the possibility to realize constructive completion during inter-model integration. This theoretical notion of model-based interoperability is likely to facilitate the tool-oriented interoperability of different modelling products in the future.

2.5 Service Models

In the following subsection service models are introduced based on examples. A human-centric and a machine-centric variant are shown.

Human-centric service models as they came up during this study are diagram-like language artefacts that sketch service instances as well as their connections. This can be done using different pictograms. In practice, such models are created at Credit Suisse by the help of MS Viso or similar tools. In the following, we will always assume that such diagram languages are given by a catalogue of icons and a corresponding graph grammar. However, the assumption is that both, the catalogue of language icons and the graph grammar can be changed. In detail, new elements can be added to the catalogue, old ones can be deleted or modified, as well as new graph transformation rules can be added to the graph grammar and old ones can be deleted or modified. Therefore, a diagram-like domain language for service landscapes can be kept open as it was requested by people at Credit Suisse to handle new and possibly unforeseen situations.

Because business and IT people are assumed to model their service landscapes independently we do give two examples for human-centric service models in the following. Both models can be aligned to document how business services are realized by the help of IT services, or how IT services drive business services. This alignment can be realized by the help of model integration techniques as it will be shown in Sec. 4.

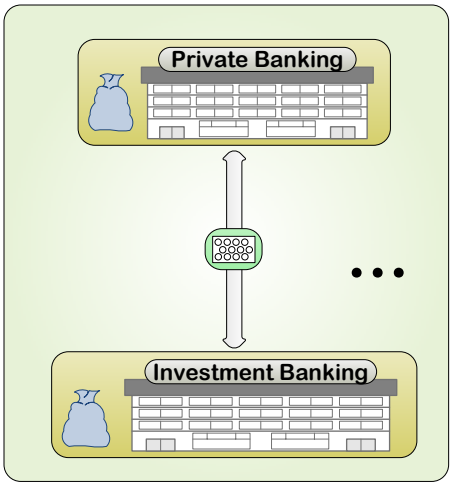


Figure 6: Human-centric Model of Business Services

Example 1 (Human-centric Model for a Business Service Structure)**.** *A fragment of a human-centric business service model is shown in Fig. 6. The departments "Investment Banking" and "Private Banking" are departments at Credit Suisse. Information exchanged between both parties must comply to the Chinese Wall Policy [16]. The policy defines what information is allowed to be exchanged between both departments. To guarantee that the policy is respected a filter will suppress illegal messages between both service instances in the diagram. Each service instance represents a department. Therefore, the policy is realized as a filter in a service model. This business view completely abstracts away IT details.*

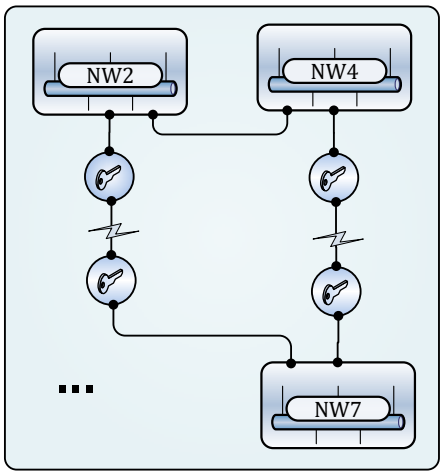


Figure 7: Human-centric Model of IT Services

Example 2 (Human-centric Model for an IT Service Structure)**.** *A fragment of a human-centric IT service model is shown in Fig. 7. Here, we see that network zone "NW4a"and "NW4b" are interconnected. We further see that network "NW4a" is connected by the help of a secured connection with network zone "NW7" and network zone "NW4a" is connected by the help of a secured connection with network zone "NW7", too.*

As a concrete alignment between the business model fragment in Fig. 6 and the IT model fragment in Fig 7, the private banking department can be mapped to network zone "NW4a" and

"NW4b" and the investment banking department can be mapped to the network zone "NW7". The connector between the investment and the banking department in the business universe is then related with the connections between network "NW4a" and "NW7" as well as "NW4b" and "NW7". Therefore, this relation is not necessarily a one-to-one mapping.

Human-centric service models as they have just been introduced are only syntax artefacts. A corresponding semantics can be assigned by the help of an alignment with machine-centric models. The corresponding type of machine-centric model that is used here are abstract behaviour types and Reo connectors [11, 17, 18, 19]. This kind of model is based on formal methods. Its set of elementary icons and its graph grammar are fix.

The reason why we propose to use abstract behaviour types and reo connectors to specify services and service landscapes is because of their support for exogenous coordination. In addition to that abstract behavior types focus on incoming and outgoing messages and, therefore, abstract away implementation details of services which frees an ABT-Reo model from implementation aspects and reduces the overall complexity of models.

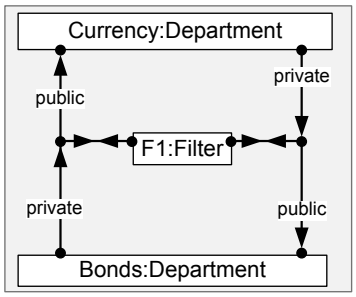


Figure 8: ABT-Reo Instance in the Business Universe

Example 3 (ABT-Reo Instance)**.** *A fragment of machine-centric business service model is shown in its concrete syntax in Fig. 8. Here, two abstract behaviour types are used to represent the investment and the private banking department. Messages running between these two abstract behaviour types have to pass through different Reo connectors. Messages using private connectors are not visible to the outside. Messages using public connectors are visible. The filter in the middle of the diagram listens to messages and will suppress private messages that are trying to get on a public connector. This fragment of a machine-centric business service model is able to formally specify by the help of formal methods organizational security policies like the Chinese Wall Policy. Using model-checking it is possible to prove that no private message will finally pass by a public connector.*

Besides the modelling of business service structures ABT-Reo diagrams can also be used to model IT service structures. Both types are intended to be aligned with their corresponding conceptual service models and in this way the human centric models shall be formally analysed using the ABT-Reo diagrams.

2.6 Modelling of Processes and Organizational Rules

In order to specify possible workflows within a concrete enterprise and to formalize the given policies and laws the presented model framework shows separate dimensions for these aspects. This paper has a special focus to service models and we describe possible techniques for the processes and rules briefly. Accordingly, the Sections 3 and 4 for intra- and inter-model techniques focus on service models.

2.6.1 Event Driven Process Chains

Event driven process chains (EPCs) [20] are a common modelling language for business processes. It is preferred by many modelers, because they like the intuitive notation, which is easy to grasp and to understand. EPCs have been extended to suit different needs and in the case of Credit Suisse, there is a special interest to have a support for the generation of continuity processes and the analysis of non-functional requirements, e.g. security constraints. For this purpose, the language of WDEPCs (data-flow oriented EPCs from the point of view of a workflow engine) was introduced in [12]. These enterprise models for business processes build a basis for process analysis with respect to non-functional requirements and furthermore, for the generation of continuity processes based on the standard process that are equipped with process fragments for specific failures of the system components. The generated processes are ensured to be executable, produce at least the same or equivalent output as the standard process and some of the non-functional requirements are checked to be fulfilled.

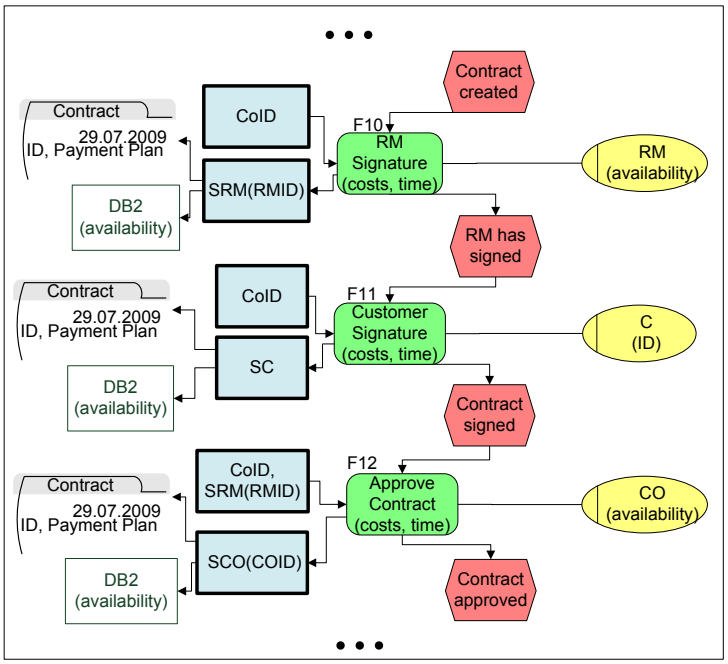


Figure 9: Part of a Workflow Model

Example 4 (Business Process Model)**.** *Figure 9 shows a part of a WDEPC-business process model for a loan granting process. The figure shows three steps, which are situated in the middle of the overall process. The three steps describe the three signatures that are placed on the loan contract by the involved parties. The signatures are placed by the relationship manager (RM, function F10) that serves the customer, by the customer himself (C, function F11) and finally, by the credit officer (CO, function F12) in order to complete the contract.*

The system components that are involved in the process can fail and the actors can be unavailable. Both is specified by the term "availability". For a concrete combination of failures alternative continuations of the process have to be processed that respect the functional and non-functional requirements of the process. For this purpose, the model is formalized by a graph grammar, in which the effect of each function is specified by a rule. The grammar is constructed automatically and supports the analysis of dependencies between the steps using the well founded results for graph transformation.

The validation of non-functional requirements is performed by graph constraint checks, e.g. the constraint in Fig. 10 specifies a requirement for the creation of a contract according to the

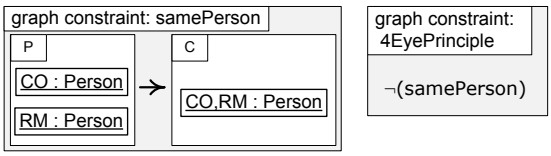


Figure 10: Graph Constraint 4-eye principle

four-eye principle . It ensures that the employee roles relationship manager ("RM") and credit officer ("CO") are performed by two different persons in a concrete execution of the workflow schema. The constraint "4EyePrinciple" is a negation of the constraint "samePerson" and thus, it requires that the constraint "samePerson", i.e. the two roles are not assigned to a single person in a concrete execution. Further details of graph constraints are subject of Sections 3.3 and 4.4, where we apply this technique to specify requirements for the structure of service models. The complete process model, its analysis and the generation of continuity processes is described in [12].

2.6.2 Process Algebra mCRL2

In the machine centric dimension of process models we propose to use the language mCRL2 (micro Common Representation Language 2) [10], which extends the process algebra ACP (Algebra of Communicating Processes) by data and time aspects. In [21], the mCRL2 tool-set is used for an analysis of a loan granting process that is similar to the one in Sec. 2.6.1. Figure 11 shows the specification of the four-eye principle with respect to parallel executions of the workflow. This time, there is no explicit distinction between the credit officer and the relationship manager - both have the role of a relationship manager. Nevertheless, the tool-set for mCRL2 allows to validate that the computed state space for parallel executions does not contain a path, in which the four-eye principle is violated for the functions "F10_RM_Signature" and "F12_Approve_Contract" with respect to the workflow model in Fig. 10.

```
1 forall e:RelationshipManager, c:Customer.
2   [true*.F10_RM_Signature(e,c).true*.
3   F12_Approve_Contranct(e,c)]false
```

Figure 11: mCRL2 Specification of the segregation of duty (4-eye-principle)

The tool-set allows the modeler to specify many detailed requirements based on modal logic. This enables, e.g. the specification of requirements regarding data flow. In particular, Credit Suisse has to ensure the requirement that balance and address information of a customer must not be send at the same time.

The alignment between the human- and machine-centric process models, i.e. between WDEPCs and mCRL2 specifications, shall be based on their abstract syntax graphs. This will allow to reflect analysis results from the machine-centric models to the intuitive human centric models. The inter-model techniques that we use in this paper are general with respect to domain languages and there is a potential that they can be used to establish alignments not only for service models but also for process models.

Besides the process algebra mCRL2, there are also other machine-centric modelling techniques, especially Petri nets [22], which have shown to be suitable for business workflows [23] and several tool-sets have been implemented.

2.6.3 Organizational Rules

In order to ensure security, risk and compliance requirements given by policy rules there is a need for a formal analysis of the models within the enterprise model framework in Fig. 5. There are

different kinds of organizational rules, which are formulated by several organizations as well as by the enterprise itself. A formalization can be performed in a first step using a specified subset of a natural language for a specific business domain. This way, the semantics and structure of the rules given as sentences can be restricted and equipped with an abstract syntax and these language artefacts are still human-centric.

In the machine-centric dimension, there is a good potential to completely formalize many of the rules by first order logic (FOL). This way, the rules specified by formulas can be evaluated and checked using rule-engines as there are for example Prolog-based rule-processing implementations available. Furthermore, there is a close relationship between first order logic and graph constraints [24], such that the analysis of several formulas can be transferred to graph constraint checks on the abstract syntax graphs. The alignment between the human-centric policy rules and the formulas in first order logic shall be based on their abstract syntax graphs. Note that this alignment is usually not one-to-one, i.e. there may be several sentences in a domain language that correspond to a single formula and vice versa. A flexible alignment is a special need in the context of Credit Suisse, because such domain languages need to be extendable while first order logic keeps fixed.

Furthermore, in order to formalize and analyze some of the requirements that generally specify the permitted workflows, graph transformation systems can be used. They have shown to be a practicable and intuitive concept for the specification of the operational semantics of workflow models, including extensions for data and control flow [25].

2.7 Scope of the Paper within the Model Framework

The enterprise model framework as shown in Fig. 5 encompasses the development of many different aspects. This paper presents suitable intra- and and inter-modelling techniques focussed on machine-centric business and IT service models given by ABT-Reo diagrams. The techniques are general with respect to different domain specific languages, because they are based on the underlying abstract syntax graphs of the models. Thus, there is a good potential that they can be applied for several other dimensions in the enterprise model framework, too.

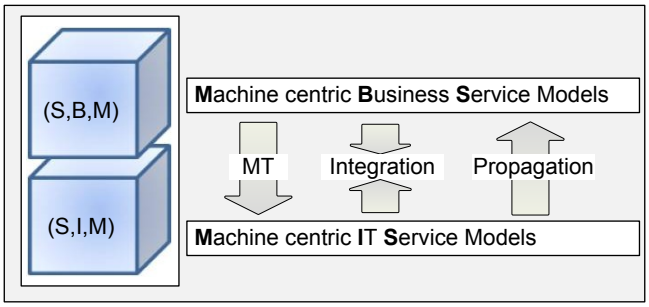


Figure 12: Scope of the Paper

The scope of this paper for intra- and inter-model techniques in the following Sections 3 and 4 is illustrated in Fig. 12 and the examples are based on our previous work in [26]. Algebraic graph transformation is applied for both, intra- and inter-modelling. It is used to specify the construction of the abstract syntax graphs during the model development and the analysis of functional and non-functional requirements. The inter-model techniques are based on triple graphs, triple graph transformation and triple graph grammars [27]. We present how model transformation is used to transform a business service model into an IT service model. This way, model transformation supports the modelling when certain models in one domain are present, but their counterparts in a connected domain are missing. Furthermore, we show how business and IT service models can be integrated, i.e. how the relations between their elements can be automatically established and how inconsistencies can be detected. Finally, we show how the propagation of graph constraints can enable the transfer of especially non-functional requirements from the IT domain to the business

domain of service models.

3 Intra-Model Techniques

The enterprise model framework presented in Sec. 2 captures various kinds of models and the development of these models is performed on its concrete syntax, i.e. in visual notation. Synchronously to the construction of the concrete syntax a development environment creates the underlying abstract syntax elements which form abstract syntax graphs as described e.g. in [28]. Hence, the structural and formal analysis of these models can be based on the underlying formal abstract graph structure.

This section presents suitable techniques for the formal and automated construction and for the analysis of the created abstract syntax graphs. The benefits of the analysis are the following. First of all, there is a user support for the detection and correction of violations against well-formedness rules with respect to models. The detection and highlighting of errors is performed automatically and the modeler can chose whether he corrects the model manually or starts an automatic correction process. In addition to that, a substantial part of the given functional and nonfunctional requirements for a model can be formulated in an intuitive and visual way using graph constraints as formal technique. These graph constraints can be checked automatically based on the underlying formal abstract syntax and the checks can be implemented in the development tools. The formal framework for these techniques is algebraic graph transformation [5].

Unlike the standard approach in graph transformation in which a graph language is defined by one graph grammar we define two separate graph grammars: a construction and a correction graph grammar (C & C). The construction grammar contains compact and general editing rules that can be attached to the editing operations of a model development environment. The correction grammar is used for the detection and elimination of incomplete respectively inconsistent structures.

This way the modelling process is kept flexible, i.e. the modeler is able to freely edit the models implying that models may violate language constraints at intermediate steps. Furthermore, the visual modelling languages are not restricted in the way visual elements are aligned. Each visualization can be based on abstract syntax graphs, where each node type can be visualized by a visual element in the concrete syntax.

Because of the need for flexibility during the editing process, the construction grammar usually allows to generate invalid models. But the correction grammar contains rules that detect inconsistent respectively incomplete parts of models in order to correct them. Those incorrect parts can be highlighted in a modelling environment, such that the modeler has two options for correction. Either he modifies the problematic model parts himself or he starts the cleaning up process, in which the problematic elements are deleted by an automated application of the rules of the correction grammar. In order to validate the soundness of the automated correction we can analyze confluence by applying central results proven for graph transformation systems in general. In addition to the analysis of the modes with respect to well-formedness rules we present a structural analysis of the abstract syntax graphs based on graph constraints that are used to verify functional an non-functional constraints, such as security requirements or modelling guidelines. If a constraint is violated the execution of the graph constraint checks detects the problematic parts of the models.

An adequate set of models $M_{X,Y,Z} \in \mathcal{M}_{X,Y,Z}$ in the coordinate (X, Y, Z) in the enterprise model framework in Fig. 5 is obtained by the combination of the three techniques explained above and applied in the following way. The models are generated using the construction grammar and a correction using the correction grammar thereafter for eliminating the models that are not well-formed. A further restriction is obtained using the graph constraint checks for excluding models that do not fulfill the functional and non-functional requirements. In this section we concentrate on the service models in the coordinates (S, B, M) and (S, I, M) and we illustrate the three techniques by representative rules and an intuitive constraint.

The following Sec. 3.1 describes some basic notions for the formal foundations of graph gram-

mars and presents parts of the construction grammar $GG_{CON-ABT}$ for the language of ABT-Reo diagrams for business and IT service models. Thereafter, Sec. 3.2 presents the grammar $GG_{COR-ABT}$ for the correction of intermediate models constructed by $GG_{CON-ABT}$, where we use the concept of negative application conditions for the detailed specification of critical patterns. The complete grammars are given in [6]. Finally, Sec. 3.3 shows how graph constraints are used to specify and check structural requirements that capture some of the functional and non-functional aspects and in Sec. 3.4 we summarize the achievements of this section.

3.1 Construction of Models by Construction Graph Grammars

The framework of algebraic graph transformation provides a formal foundation for the specification and modification of object oriented models, in which the concrete syntax of these models is related to their abstract syntax graphs. In order to present representative parts of the construction grammar $GG_{CON-ABT}$ for ABT-Reo diagrams this section reviews the general notions of typed graph transformation. For details about attribution and type graphs with inheritance we refer to [5].

The algebraic approach of graph transformation uses the notion of directed graphs with explicit functions that point to the source and target nodes of an edge. Mappings between graphs are given by graph morphisms that are compatible with the internal structure of the graphs, i.e. the source node of an edge in the domain graph is mapped to the source node of the mapped edge in the image graph (and similarly for the target nodes).

Definition 1 (Graphs and Graph Morphisms)**.** *A graph $G = (V_G, E_G, s_G, t_G)$ consists of a set V_G of nodes, a set E_G of edges, and two functions $s_G, t_G : E_G \to V_G$, the source and the target function.*

Given two graphs G and H a graph morphism $f : G \to H$, $f = (f_V, f_E)$ consists of two functions $f_V : V_G \to V_H$ and $f_E : E_G \to E_H$ that preserve the source and the target functions, i.e. $f_V \circ s_G = s_H \circ f_E$ and $f_V \circ t_G = t_H \circ f_E$. Graphs and graph morphisms define the category **Graphs**. *A graph morphism f is injective if both functions f_V, f_E are injective.*

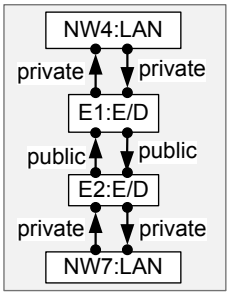


Figure 13: ABT-Reo Instance M_1 of the IT Universe

Example 5 (Graph)**.** *Figure 13 shows the ABT-Reo diagram M_1 in concrete syntax specifying the structure of a part of a network composed of local area networks (LANs). It contains four ABT elements that are connected via Reo connectors. The two outer ABT elements represent the LANs "NW4" and "NW7" while the two inner ones denote encryption/decryption nodes, i.e. the communication between both LANs is encrypted.*

A part of the model M_1 is shown in 14 in both, concrete and abstract syntax. Bold bullets in concrete syntax correspond to nodes of type "Point" in abstract syntax and arrows correspond to Reo connectors, i.e. nodes of type "Reo" in the abstract syntax graph. They are attached to external input resp. external output ports according to the direction of the Reo connectors. Each point glues one input with one output port, e.g. the left Reo connector in concrete syntax corresponds to the left Reo node in the abstract syntax graph and the communication data enters the connector via

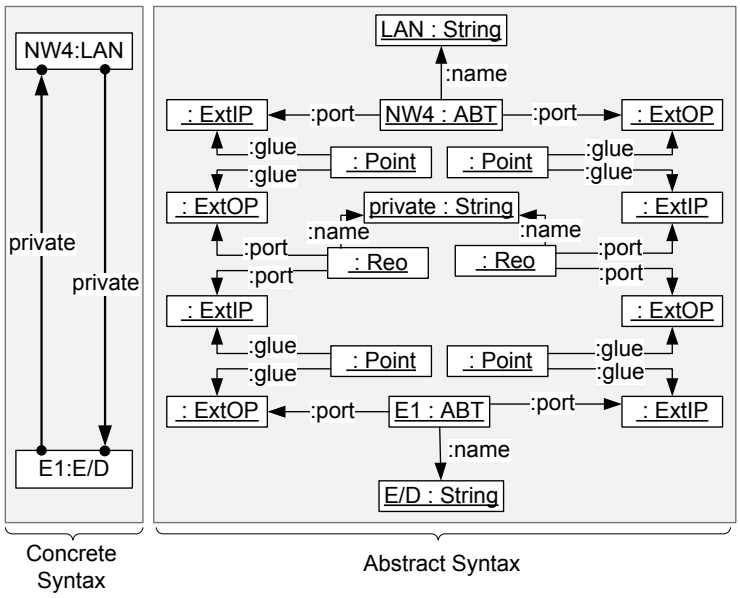


Figure 14: Part of M_1 in Fig. 13 in Concrete and Abstract Syntax

the input port at the bottom and exits the connector via the output port at the top. The type graph for ABT-Reo diagrams is shown in Fig. 15 and described in Ex. 6.

While the concrete syntax of ABT-Reo models is more compact and intuitive, a precise and detailed specification and analysis is based on the abstract syntax, which enables e.g. to explicitly specify properties of ports that are only implicit in the concrete notion.

Similar to the definition of a meta model of a visual language according to the OMG MOF approach [29] a type graph specifies the general structure in the graph grammar approach. Graphs of the language are typed over its type graph via a graph morphism that maps each element to its type element in the type graph, i.e. each node to a node and each edge to an edge in the type graph.

Definition 2 (Type Graph)**.** *A type graph is a distinguished graph TG. A tuple* $(G, type_G)$ *of a graph* G *together with a graph morphism* $type_G : G \to TG$ *is called a typed graph. Given typed graphs* $G = (G, type_G)$ *and* $H = (H, type_H)$*, a typed graph morphism* f *is a graph morphism* $f : G \to H$*, such that* $type_H \circ f = type_G$*.*

Example 6 (Type Graph)**.** *The structure of ABT-Reo diagrams in abstract syntax is given by the type graph* $TG_{ABT-Reo}$ *in Fig. 15 containing the main types "ABT" for abstract behaviour type nodes, "Reo" for Reo connectors, "Port" for ports and "Point" for points that glue together input and output ports of both, ABT nodes and Reo connectors. Ports of elementary ABT nodes and Reo connectors are external, i.e. they are used for external communication with other elements. ABT nodes can also be composite, i.e. they contain a further specified internal structure involving other ABT nodes and Reo connectors. In this case their external ports are connected to complementary internal ports, such that the communication is transferred through the borders of the composite ABT nodes.*

General editing steps of graph based models are specified by graph transformation rules. The left hand side (LHS) of a rule defines the pattern that has to be found in a graph before applying the rule and it is replaced by the right hand side (RHS) of the rule during the rule application. Both sides of a rule are related by an intermediate graph K, which defines the elements that are preserved during an application of the rule. Since the graph K is implicitly given by the LHS and

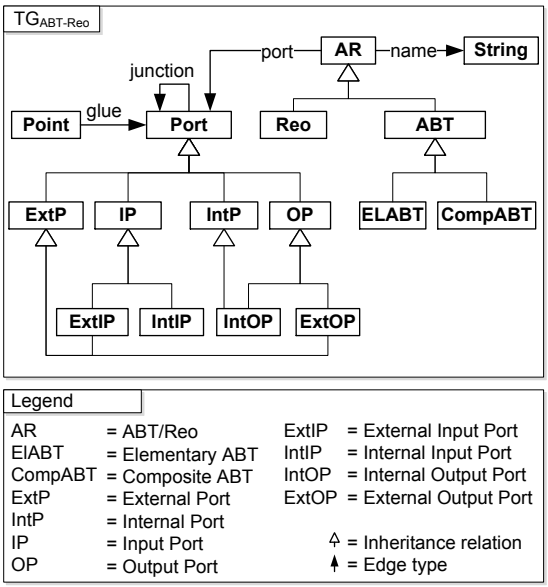


Figure 15: Type Graph $TG_{ABT-Reo}$ for ABT-Reo models

RHS of a rule we will usually omit this graph in the figures and denote the mappings between the LHS and the RHS of a rule by numbers.

A rule may further contain a negative application condition (NAC) that specifies forbidden patterns. The effect is that the rule is not applicable if the NAC pattern is present at the matched part of the graph. This improves the detailed specification of rules. A few rules in the construction grammar contain NACs, in order to prevent editing steps that are not adequate in general. This improves the model quality before checking the well-formedness conditions. However, the NACs can also be ommited in the construction grammar to ensure maximal flexibility for the modeler, but then the correction grammar becomes more complex. In any case, NACs are very important for the correction grammar to specify incomplete patterns, i.e. the NACs specify the parts that are missing as described in Sec. 3.2.

Definition 3 (Typed Graph Rule)**.** *A typed graph rule* $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ *consists of typed graphs* L, K*, and* R*, called the left-hand side, gluing graph, and the right-hand side respectively, and two injective typed graph morphisms* l *and* r*. A rule may furthermore be equipped with additional NACs defining forbidden structures. A NAC of* p *is given by a graph* N *together with an injective graph morphism* $n : L \to N$*.*

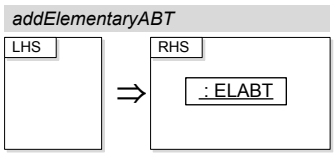


Figure 16: Rule "addElementaryABT" of $GG_{CON-ABT}$

Some editing steps can be more complex in the way that they combine the effect of basic editing steps. In this case the complex editing step combines the effect of several basic construction rules and the rules needed for this complex step are combined leading to a new rule. The combination of rules is performed by a form of gluing via a common subpart E, which is formalized by the construction of E-concurrent rules in [5].

```
addElementaryABT$L = makeTypedGraph[  (* === L ===*)
   {}, {}, AR$TG];
addElementaryABT$R = makeTypedGraph[  (* === R ===*)
   { {rv1, AR$ElementaryABT} },
   {}, AR$TG];
(* Rule L <- K -> R *)
addElementaryABT = makeRule[addElementaryABT$L, addElementaryABT$R, {}];
```

Figure 17: Source Code for Rule addElementaryABT

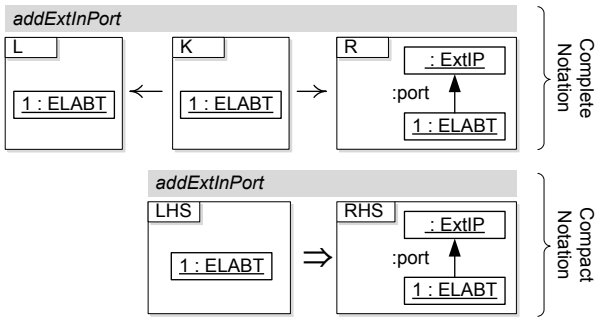


Figure 18: Rule "addExtInPort" of $GG_{CON-ABT}$

```
addInPort$L = makeTypedGraph[  (* === L ===*)
   { {n1, AR$ElementaryABT} },
   { },
   AR$TG];
addInPort$R = makeTypedGraph[  (* === R ===*)
   { {n1, AR$ElementaryABT}, {rv1, AR$ExternalInputPort} },
   { {re1, {n1, rv1}, AR$ABTPort} },
   AR$TG];
(*Rule L<-K->R*)
addInPort = makeRule[addInPort$L, addInPort$R, {}];
```

Figure 19: Source Code for Rule "addExtInPort"

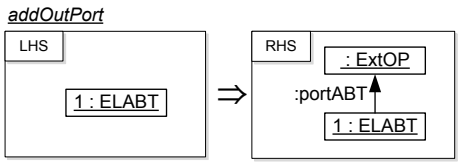


Figure 20: Rule "addExtOutPort" of $GG_{CON-ABT}$

The rules in Figures 18 to 26 are part of the construction grammar $GG_{CON-ABT}$ for the creation of ABT-Reo diagrams. Each editing step of a development environment can be equipped with one or more corresponding graph rules depending on the complexity of the editing step. The combination of rules is performed by a form of gluing via a common subpart E, which is formalised by the construction of E-concurrent rules in [5]. Rule "addExtInPort" in Fig. 18 specifies the creation of a new external input port that is attached to an existing elementary ABT node, i.e. an ABT element without internal structure. The rule is shown in complete notation including the intermediate graph K and in compact notation leaving K implicit. In compact notation the graph K is given by the numbered elements, i.e. the elements that occur in the LHS and the

```
addExtOutPort$L = makeTypedGraph[  (* === L ===*)
   { {n1, AR$ElementaryABT} },
   { },
   AR$TG];
addExtOutPort$R = makeTypedGraph[  (* === R ===*)
   { {n1, AR$ElementaryABT}, {rv1, AR$ExternalOutputPort} },
   { {re1, {n1, rv1}, AR$ABTPort} },
   AR$TG];
(* Rule L<-K->R *)
addExtOutPort = makeRule[addExtOutPort$L, addExtOutPort$R, {}];
```

Figure 21: Source Code for Rule addExtOutPort

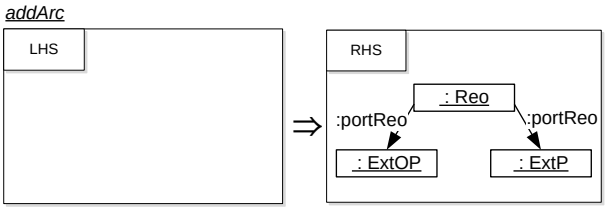


Figure 22: Rule addArc

```
addArc$L = makeTypedGraph[  (* === L ===*)
   {}, {}, AR$TG];
addArc$R = makeTypedGraph[  (* === R ===*)
   { {rv1, AR$Reo},
     {rv2, AR$ExternalOutputPort},
     {rv3, AR$ExternalInputPort}
    },
   { {re1, {rv1, rv2}, AR$ReoPort},
     {re2, {rv1, rv3}, AR$ReoPort}
    }, AR$TG];
(* Rule L<-K->R *)
addArc = makeRule[addArc$L, addArc$R, {}];
```

Figure 23: Source Code for Rule addArc

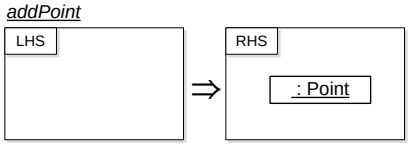


Figure 24: Rule addPoint

RHS of a rule. Since the rule "addExtInPort" does not delete any elements the graphs L and K are identical. In the case of a deleting rule the graph L contains the graph K and additionally the elements that are deleted by the rule.

The rule "addElementaryABT" in Fig. 16 has an empty LHS and thus, it can be applied without any precondition. The effect of the rule is the creation of a new elementary ABT. Fig. 26 shows the rule "glue" which specifies the gluing of ports. In order to glue together two ports via a point (a bold bullet in the concrete syntax) the rule is applied twice to the same point but to different ports. The negative application condition "NAC1" is equal to the RHS of the rule

```
addPoint$L = makeTypedGraph[   (* === L ===*)
   {}, {}, AR$TG];
addPoint$R = makeTypedGraph[   (* === R ===*)
   { {rv1, AR$Point} },
   {}, AR$TG];
(*Rule L<-K->R*)
addPoint = makeRule[addPoint$L, addPoint$R, {}];
```

Figure 25: Source Code for Rule addPoint

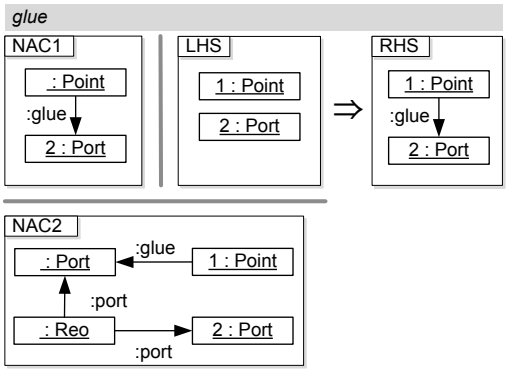


Figure 26: Rule "glue" of $GG_{CON-ABT}$

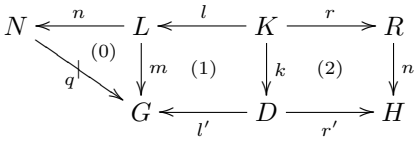
and thus, it forbids a repeated application of the rule "glue" at the same match, i.e. if the point "1" is already connected to the port "2" then the rule cannot be applied at these nodes. The second NAC "NAC2" ensures that no two ports of a Reo-connector are directly connected with each other. Note that the rule extensively uses the inheritance structure in the type graph in the way that the rule is applicable for any type of ports, i.e. for external, internal, input and for output ports.

Formally, a graph transformation step from G to H via a rule p and a match m is defined by two pushout diagrams (1) and (2) as shown in Def. 4. For the formal definition of pushouts we refer to [5]. The main ideas are the following. Given a match of the LHS of the rule into a graph G then this concrete part is replaced by the RHS of the rule if the rule is applicable, which requires that the gluing condition is satisfied and the match is NAC consistent. A negative application condition (NAC) specifies a negative pattern that must not occur at the match when applying the rule. The gluing condition requires that the identification and dangling points are gluing points, i.e. that they are preserved and thus belong to the set $GP = l_V(V_k) \cup l_E(E_K)$. The identification points are those nodes and edges that are matched non-injectively in G, i.e. at least two elements are mapped to the same element in G. The dangling points specify those nodes which are matched to nodes in G, such that there is an adjacent edge which is not deleted by the rule. This means that the edge would remain dangling by removing the node. Thus, an identification or a dangling point is never deleted. The gluing condition ensures the existence of the intermediate object D and it has to be checked only for deleting rules.

If a rule p is applicable to G via $m : L \to G$ then the graph transformation step $G \overset{p}{\Rightarrow} H$ is performed by first deleting all elements in G that are matched by the rule but not preserved, i.e. they do not occur in the intersection K of the LHS and RHS of the rule. The deletion leads to an intermediate graph D. The second step is performed by adding those elements to D that are in the RHS of the rule but not in K leading to the resulting graph H.

Definition 4 (Typed Graph Transformation Step)**.** *Given a rule $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ with NACs and a match $m : L \to G$ then p is applicable to G via m if the gluing condition is satisfied and the match is NAC consistent. The match is NAC consistent, if for each NAC $n : L \to N$ of p there is*

no injective $q : N \to G$ compatible with m, i.e. with $q \circ n = m$ as shown in the triangle diagram (0). *Given a rule p that is applicable to G via m, then the graph transformation step $G \overset{p,m}{\Longrightarrow} H$ is defined by the two pushouts* (1) *and* (2).



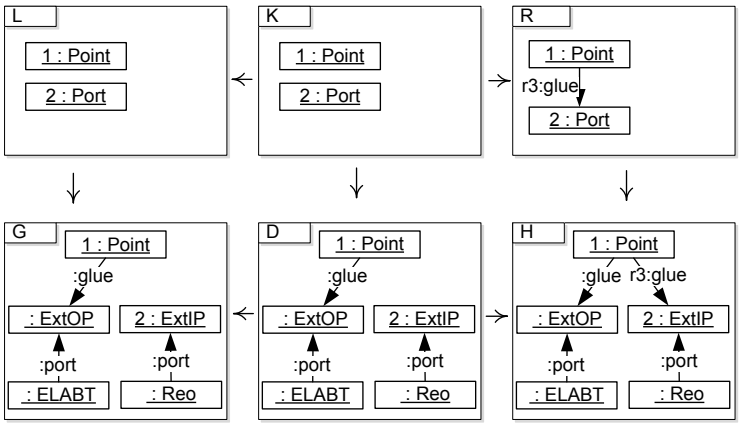


Figure 27: Application of Rule "glue"

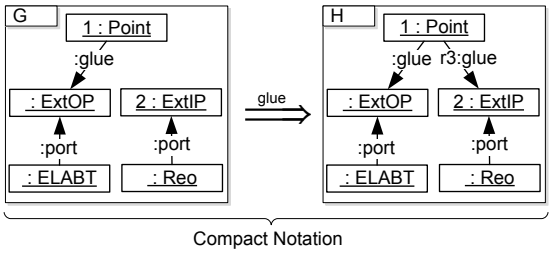


Figure 28: Application of Rule "glue" in Compact Notation

Example 7 (Typed Graph Transformation Step)**.** *The graph transformation step in Fig. 27 shows the application of the rule "glue" at a graph G, which contains an elementary ABT node with an external output port and a Reo connector with an external input port. The match m is denoted by the numbers "1" and "2". The rule is applicable, because no node is deleted and the NACs are fulfilled, i.e. nodes "1" and "2" are not connected already and the involved ports do not belong to the same Reo connector. In a first step we construct D as subgraph of G, where all elements of $L \setminus K$ are deleted. In our case we have $G = D$, because $L = K$. In a second step we glue together graphs D and R via K resulting in graph H. This gluing construction is formally given by a pushout in the category of graphs [5]. In fact, both diagrams* (1) *and* (2) *in Fig. 27 are pushouts leading to a transformation step $G \overset{p,m}{\Longrightarrow} H$ via rule $p = glue$ and match m. Figure 28 shows the transformation step in compact notation leaving the DPO diagram implicit.*

A graph grammar consists of a type graph together with a typed start graph and a set of typed rules. The language generated by this grammar consists of all typed graphs that can be obtained by applying a sequence of rules to the start graph.

Definition 5 (Typed Graph Grammar). *A typed graph grammar $GG = (TG, SG, P)$ consists of a type graph TG, a start graph SG and a set of graph rules P also called productions, both typed over TG. The typed graph language $L(GG)$ of GG is defined by $L(GG) = \{G \mid \exists$ typed graph transformation $SG \Rightarrow^* G$ in $GG\}$.*

Example 8 (Construction Grammar). *The graph grammar $GG_{CON-ABT} = (TG_{ABT-Reo}, \emptyset, P_{CON})$ consists of the type graph shown in Fig. 15, the empty graph as start graph and the set P_{CON} of construction rules. Some of the rules of P_{CON} are shown in Figures 18, 16 and 26. Similar to the shown rules for adding structure parts the set P_{CON} also contains the inverse rules for deleting such elements [6]. Since the rules are very compact they can be combined with general editing steps in a visual development environment, where only the concrete syntax is displayed to the modeler.*

The remaining rules of $GG_{CON-ABT}$ are shown in the following.

```
glue$L = makeTypedGraph[  (* === L === *)
   { {n1, AR$Point}, {n2, AR$Port} },
   {}, AR$TG];
glue$R = makeTypedGraph[  (* === R === *)
   { {n1, AR$Point}, {n2, AR$Port} },
   { {re1, {n1, n2}, AR$glue} },
   AR$TG];
glue$N1 = makeTypedGraph[ (* === NAC N1 === *)
   { {n1, AR$Point}, {n2, AR$Port}, {nIv1, AR$Point} },
   { {nIe1, {nIv1, n2}, AR$glue} },
   AR$TG];
glue$N2 = makeTypedGraph[ (* === NAC N2 === *)
   { {n1, AR$Point}, {n2, AR$Port}, {nIIv1, AR$Port}, {nIIv2, AR$Reo} },
   { {nIIe1, {nIIv2, nIIv1}, AR$ReoPort},
     {nIIe2, {nIIv1, n1}, AR$glue},
     {nIIe3, {nIIv2, n2}, AR$ReoPort} },
   AR$TG];
(* Rule L <- K -> R *)
glue = makeRule[glue$L, glue$R, {{"NAC", glue$N1}, {"NAC", glue$N2}}];
```

Figure 29: Source Code for Rule glue

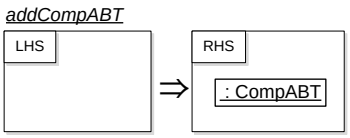


Figure 30: Rule addCompABT

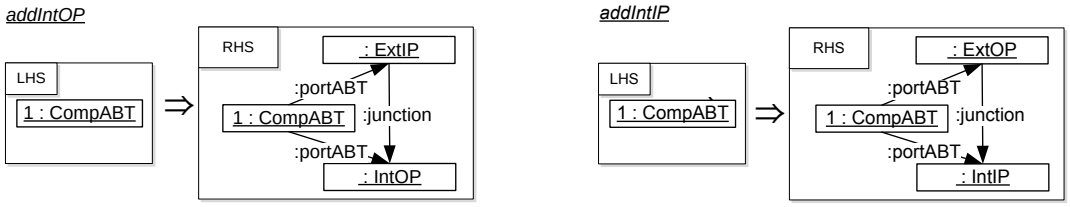


Figure 31: Rules addIntOP and addIntIP

Attribution rules Special: attribution in instance models with attributes of generic type "Value".

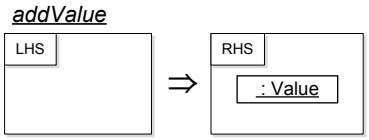


Figure 32: Rule addValue

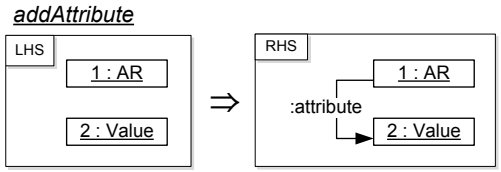


Figure 33: Rule addAttribute

update and extend description

3.2 Correction of Non-Well-formed Models by Correction Graph Grammars

During the development of ABT-Reo models in a visual editor - based on the grammar $GG_{CON-ABT}$ from the previous section - the edited models may not satisfy the well-formedness rules of the language of ABT-Reo diagrams defined in [11]. Therefore, this section presents how a second grammar $GG_{COR-ABT}$ is used to detect and highlight incomplete as well as incorrect parts, such that they can be deleted manually or corrected automatically depending on the preference of the modeler.

More precisely, the set of correction rules P_{COR} of the correction grammar $GG_{COR-ABT}$ is used for the detection and correction of non-well-formed patterns. The detection is performed by checking the applicability of the rules of $GG_{COR-ABT}$. If a rule is applicable, then the found match specifies the location of the problematic pattern and the rule describes the type of the occurred problem. The automated correction is performed by applying the correction rule. An overall correction is obtained by applying the correction rules as long as possible. As an alternative, the modeler can choose to correct some of the highlighted parts himself.

Using the construction and the correction grammar we intend to be able to construct all well-formed ABT-Reo diagrams - and similar other well-formed diagrams - in the following way. We first use the editing rules of the construction grammar to construct a transformation sequence $\emptyset \Rightarrow G_1 \Rightarrow \ldots \Rightarrow G_n$ via the rules P_{CON} of the construction grammar. In a second step the rules P_{COR} of the correction grammar are applied as long as possible to G_n leading to a transformation sequence $G_n \Rightarrow \ldots \Rightarrow G_m$ via P_{COR}. Assuming that for each non-well-formed pattern in any graph G typed over $TG_{ABT-Reo}$ there is a corresponding rule in P_{COR} applicable to G, we can be sure that G_m is well-formed, if G_m is terminal with respect to P_{COR}. Fortunately, there are results in the algebraic theory of graph transformations assuring termination based on suitable termination criteria for typed graph grammars. A graph grammar $GG = (TG, SG, P)$ is called to be terminating, if there is no infinite transformation sequence from SG via P.

Theorem 1. *Every typed graph grammar $GG = (TG, SG, P)$ is terminating, if the termination criteria* 1 - 4 *of Theorem 3.37 in [5] are satisfied.*

Another interesting question is whether the correction process leads to a unique result independently of the order in which the correction rules are applied. The uniqueness property is valid,

if the graph grammar is confluent. Confluence is based on critical pairs, which specify conflicts in a minimal context. An important result is that confluence is assured by local confluence and termination. For local confluence we have the following result (see Thm. 3.34 in [5]).

Theorem 2. *A typed graph grammar is locally confluent if all its critical pairs are strictly confluent.*

Based on this theorem there are analysis techniques which offer sufficient conditions for local confluence. In fact, we can statically analyse whether the order in which the correction rules are applied is relevant or not for the final result of any correction process. In the case of guaranteed confluence the automated correction is always deterministic. This means that the modeler can apply any of the applicable correction rules without the risk that the correction of another fragment becomes impossible or will cause a backtracking.

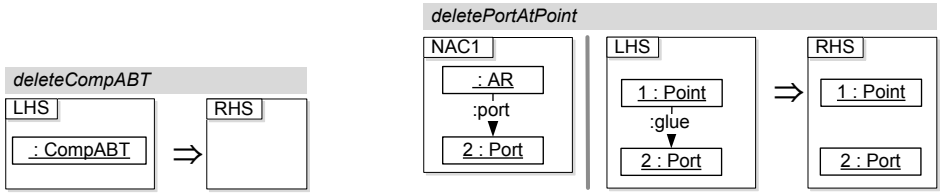


Figure 34: Rules "deleteCompABT" and "deletePortAtPoint" of $GG_{COR-ABT}$

The graph grammar $GG_{COR-ABT} = (TG_{ABT-Reo}, \emptyset, P_{COR})$ for correcting edited ABT-Reo diagrams consists of the type graph shown in Fig. 15, the empty graph as start graph and the set P_{COR} of correction rules. Two rules of P_{COR} are shown in Fig. 34. The rule "deleteCompABT" is applicable if the node of type "CompABT" is not connected to any edge, which implies that the composite ABT has no internal structure. Since composite ABT elements are required to contain some internal elements this part of the model is incomplete, which can be highlighted in the editor. Applying the rule will delete this node. The rule "deletePortAtPoint" in Fig. 34 is applicable if the port "2" is neither connected to a Reo connector nor to an ABT node. Again this incomplete pattern is detected and the edge to its connected point is deleted by the application of the rule. Another correction rule for deleting points without any adjacent edge may become applicable thereafter and in this case a new problematic pattern will be detected.

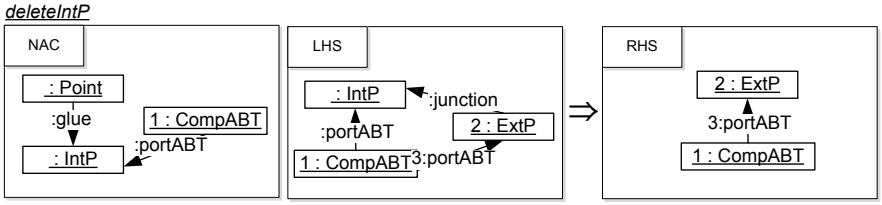


Figure 35: Rule deleteIntP

```
deleteIntPort$L = makeTypedGraph[  (* === L ===*)
   { {n1, AR$CompositeABT}, {n2, AR$ExternalPort}, {lv1, AR$InternalPort} },
   { {n3, {n1, n2}, AR$ABTPort}, {le1, {n1, lv1},
     AR$ABTPort}, {le2, {n2, lv1}, AR$junction} },
   AR$TG];
deleteIntPort$R = makeTypedGraph[  (* === R ===*)
   { {n1, AR$CompositeABT}, {n2, AR$ExternalPort} },
   { {n3, {n1, n2}, AR$ABTPort} },
   AR$TG];
deleteIntPort$N1 = makeTypedGraph[ (* === NAC N1 === *)
   { {n1, AR$CompositeABT}, {n2, AR$ExternalPort}, {nIv1, AR$InternalPort},
     {nIv2, AR$Point}, {nIv3, AR$InternalPort} },
   { {n3,     {n1, n2}, AR$ABTPort},
     {nIe1, {n1, nIv1}, AR$ABTPort},
     {nIe2, {n2, nIv1}, AR$junction},
     {nIe3, {nIv2, nIv3}, AR$glue},
     {nIe4, {n1, nIv3}, AR$ABTPort} },
   AR$TG];
(* Rule L <- K -> R *)
deleteIntPort = makeRule[deleteIntPort$L, deleteIntPort$R,
   {{"NAC", deleteIntPort$N1}}];
```

Figure 36: Source Code for Rule deleteIntPort

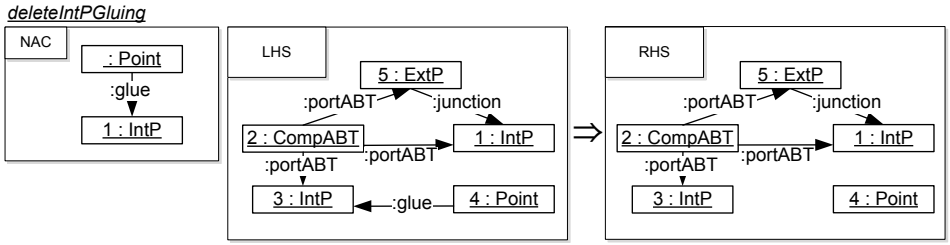


Figure 37: Rule deleteIntPGluing

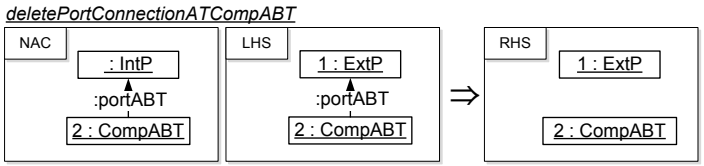


Figure 38: Rule deletePortConnectionAtCompABT

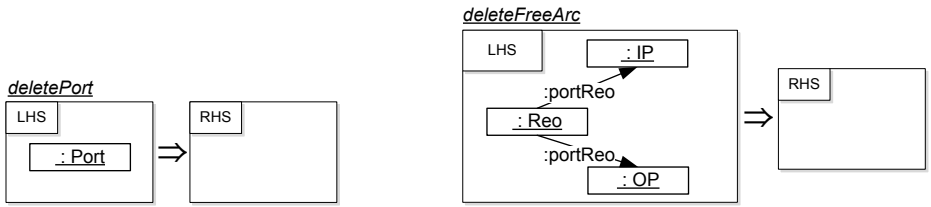


Figure 39: Rules deletePort and deleteFreeArc

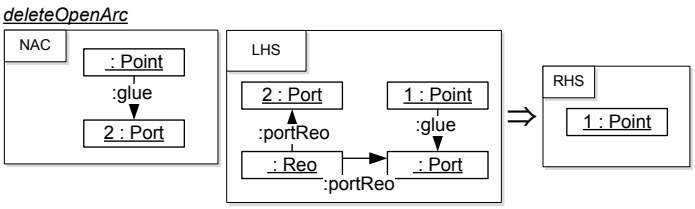


Figure 40: Rule deleteOpenArc

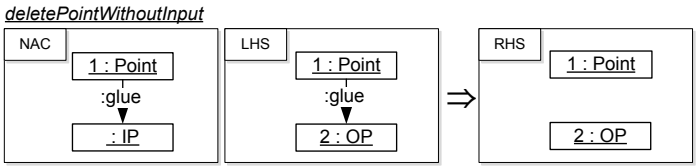


Figure 41: Rule deletePointWithoutInput

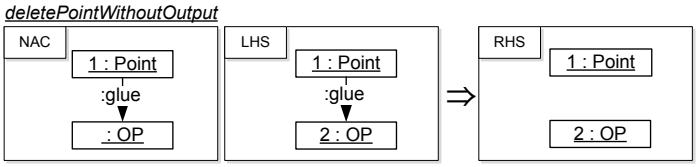


Figure 42: Rule deletePointWithoutOutput

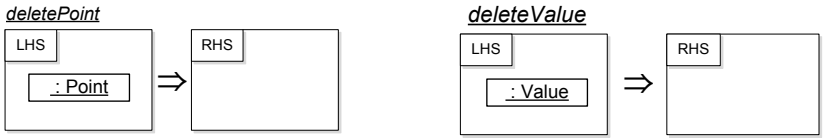


Figure 43: Rules deletePoint and deleteValue

Additional Attribute Deletion Rules Application if Value node is not connected to any node

For each new created generic attribute of a node of type t: select cleaning rule, which deletes a node of type t. Copy it and add an attribute node of type "Value" in the left and right hand side and an attribute edge in the left hand side. This way the node attribution is deleted by applying the rule. The original rule is only applicable for nodes of type t without an extra attribute.

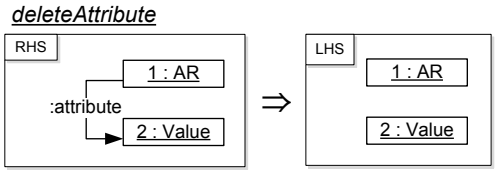


Figure 44: Schema for Rule Extension in case of Generic Attributes

- Schema for extending the cleaning rules: one rule clone as it is and the other one extended by attribute deletion

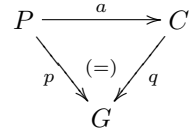
3.3 Checking Requirements by Graph Constraints

In addition to well-formedness discussed in the previous section, enterprise models have to fulfil functional and non-functional requirements, e.g. security norms, which depend on the particular domain. In order to automatically analyze and verify the compliance of models to specific requirements and norms we propose a formalization of a practicable subset of the norms and requirements by graph constraints, which show several advantages. They offer a compact and intuitive visual notation and they can be checked automatically. Since the models in the scenario of the paper are based on the abstract syntax graphs we can analyze the norms and requirements by checking the corresponding graph constraints. If needed, graph constraints can be nested leading to the full expressiveness of first order logic as shown in [24].

A graph constraint consists of a premise pattern P together with a conclusion pattern C and a morphism $a : P \to C$ that relates the elements of the premise with those of the conclusion. A graph G fulfils a graph constraint $a : P \to C$ if for any occurrence of the premise P there is also an occurrence of the conclusion C at the same position of the graph. Graph constraints can be extended to boolean formulae as specified in [5].

Definition 6 (Graph Constraint)**.** *A graph constraint* PC(a) *consists of a graph P called premise, a graph C called conclusion and a graph morphism $a : P \to C$ that relates P with C. A graph*

G fulfils PC(a) *if for any injective graph morphism* $p : P \to G$ *there is also an injective graph morphism* $q : C \to G$ *compatible with* p, *i.e.* $q \circ a = p$.



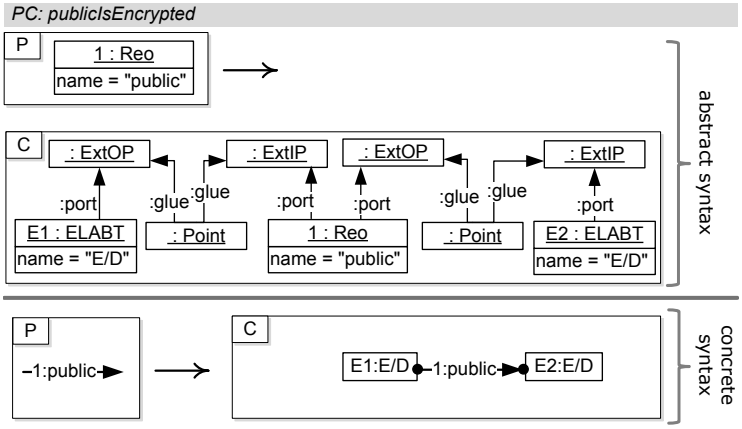


Figure 45: Graph Constraint for IT-models

```
1  (* construct graphs P and C *)
2  (* Nodes and Edges of premise graph P *)
3  PNodes={{1,ABTReo$Reo}};
4  PEdges={};
5  (* Nodes and Edges of conclusion graph C *)
6  ...
7  (* construct graphs P and C *)
8  publicIsEncryptedP=
9      makeTypedGraph[PNodes,PEdges,ABTReo$TypeGraph];
10 publicIsEncryptedC=
11     makeTypedGraph[CNodes,CEdges,ABTReo$TypeGraph];
12 (* construct constraint (P -> C) *)
13 publicIsEncrypted=makeGraphConstraint["atomic",
14     {publicIsEncryptedP,publicIsEncryptedC}];
```

Figure 46: Mathematica Source Code of "publicIsEncrypted"

```
1  In:=  checkGraphConstraint[Model1,publicIsEncrypted]
2  Out:= True
```

Figure 47: Compliance Check of a Graph Constraint

Example 9 (Graph Constraint)**.** *Figure 45 shows the graph constraint "pubclicIsEncrypted" for ABT-Reo models [11] in the IT-universe. The constraint requires that any public Reo element is connected to two ABT nodes, which implement encryption and decryption of communication data. This constraint formalizes the following verbal norm: "Confidential communication data cannot be intercepted in plain text by eavesdropping at public channels". The model* M_1 *in Fig. 13 fulfils this constraint, because for each public Reo-connector there are ABT-nodes of type "E/D" (encryption/decryption) as required by conclusion C of the constraint. Figure 46 shows the Mathematica source code for the graph constraint "publicIsEncryped" and Fig. 47 shows the operation call in Mathematica for checking the graph constraint on model* M_1 *of Fig. 13.*

If we know already that a graph G satisfies a graph constraint $PC(a)$ we would like to know under which conditions a graph transformation step $G \stackrel{p,m}{\Longrightarrow} H$ preserves this constraint, i.e. also H satisfies $PC(a)$. This would avoid to check explicitly, if H satisfies $PC(a)$. For this purpose we only have to check whether the match $m : L \to G$ satisfies a suitable application condition using the following general result for algebraic graph transformation (see Thm. 7.23 in [5]). By $A(p, a)$ we denote the the application condition for the rule p that is derived from $PC(a)$.

Theorem 3. *For each graph constraint $PC(a)$ and each production p there is an application condition $A(p, a)$ for p such that for all graph transformation steps $G \stackrel{p,m}{\Longrightarrow} H$ we have: H satisfies $PC(a)$, if G satisfies $PC(a)$ and m satisfies the application condition $A(p, a)$.*

3.4 Summary of Achievements for Intra-Modelling Techniques

In this section we have obtained the following achievements concerning intra-modelling techniques using algebraic graph transformations:

1. We have shown how to construct well-formed models, especially well-formed ABT-Reo models, using construction and correction graph grammars.

2. By Thm. 1 there are termination criteria, which ensure that after arbitrary construction steps the correction process terminates. Moreover, this process leads to a unique result independent of the order of correction steps - if according to Thm. 2 all critical pairs of the correction grammar are strictly confluent.

3. We have shown how to model and check functional and non-functional requirements by graph constraints, where Thm. 3 shows how to preserve such requirements under graph transformation steps.

After we have presented intra-modelling techniques for the construction and analysis of business and IT service models the next section presents suitable inter-modelling techniques.

4 Inter-Model Techniques

Enterprise modelling encompasses several heterogeneous aspects of the enterprise as shown in the model framework of Fig. 5. Hence, an adequate framework for enterprise modelling has to support the integration of different domain specific languages that are appropriate for the specific aspects and satisfy the preferences of the modelers. This section shows how inter-model techniques are applied in a formal and consistent way in order to ensure important properties, such as correctness and completeness. The techniques are based on the abstract syntax graphs of the models, which can be obtained as described in Sec. 3.

The application of the inter-model techniques leads to a substantial gain in model-based interoperability in the following ways. During the development process models are integrated and inconsistencies are automatically detected. Furthermore, existing models of particular source domains are transformed to models that belong to related target domains. This way the new models in the target domains can be used for further refinement in order to derive fully developed models. In addition, the new models can be checked against the non-functional requirements of the target domain. A violation of these requirements shows an inconsistency, which can be resolved by modifications of the source and the target model.

The inter-model techniques have to ensure correctness and completeness in order to produce reliable results. Therefore, we apply the well founded approach of model transformation and model integration based on triple graph grammars [27, 30, 31] for which these characteristics are shown. Besides theses formal results triple graph grammars convince by its intuitive specification of compact patterns that show how typical model fragments shall be related. Based on these patterns operational rules for model transformation and integration are derived automatically.

The following section presents the main concepts of triple graph grammars, where we intuitively describe the used techniques but refer to [30, 31] for the formal details. Thereafter Sections 4.2 to 4.4 explain the concepts of model integration, model transformation and propagation of constraints based on triple graph grammars. All concepts are illustrated by an application to business and IT service models given as ABT-Reo diagrams, i.e. an application to models in the coordinates (S, B, M) and (S, I, M) of the model framework in Fig. 5. For this reason we substantially extend the examples of [26]. Note that the techniques can be used also for source and target modelling languages that are quite different like class diagrams and relational databases as in [31], such that an application of the techniques to languages in other coordinates of the enterprise model framework shall be possible. While triple graph grammars are defined for the abstract syntax graphs of models we present their application by showing the model components in concrete syntax, i.e. by a visualization of the abstract syntax graphs as explained in [28]. This way the presentation is more compact and intuitive for this paper, but the corresponding formal theory in [30, 31] is based on abstract syntax graphs.

4.1 Inter-Modelling by Triple Graph Grammars

In the following we lift the concepts of graph transformation to the case of triple graphs, an extension of plain graphs dividing elements into source, target and correspondence sections which are connected by graph morphisms. This extension improves the definition of model transformations, where models of a source language are translated to models of a target language and correspondence elements can be used to guide the creation of the sequence of transformation steps. Similarly, triple graph transformations are suitable for model integration, which takes a source and a target model and sets up the missing correspondences between both models.

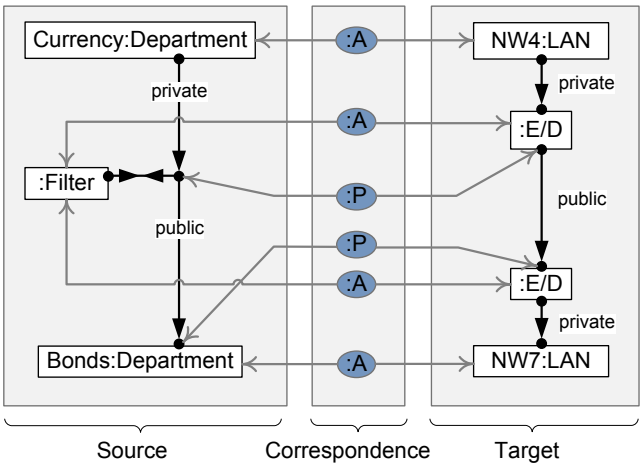


Figure 48: Triple Graph with models $M_{S,B,M}$ and $M_{S,I,M}$

Example 10 (Triple graph)**.** *The triple graph in Fig. 48 shows an integrated model consisting of a business service model in the source component (left) and an IT service model in the target component (right). Both models are ABT-Reo diagrams, i.e. they contain abstract behaviour type nodes that are connected by Reo connectors. The source model specifies that the data in the communication channels between the private banking and investment banking departments is filtered. The filter has to ensure that the communicated data does not contain files which contain both, address and balance information. The target model on the other side specifies the IT service structure. The local area networks "NW4" and "NW7", which are used in the private banking and investment banking departments, are connected via an encrypted communication channel shown by the ABT nodes of type "E/D" for encryption and decryption. The corresponding elements of both*

models are related by graph morphisms (indicated in grey) from the correspondence graph (light blue) to source and target, respectively.

Model transformation as well as model integration do not require deletion during the transformation. The result of a model integration is the triple graph of the triple transformation sequence. In the case of model transformations the result is obtained by restricting the resulting triple graph to its target component. Thus it is sufficient to consider triple rules that are non-deleting. This implies that the first step in the DPO graph transformation approach can be omitted, because the creation of elements is performed in the second step.

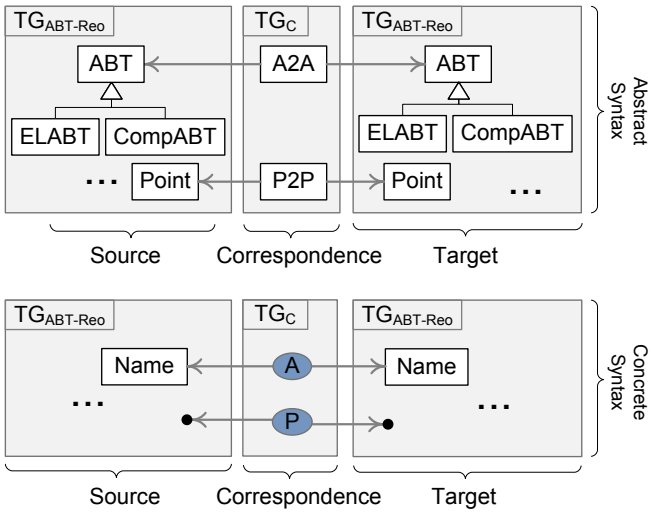


Figure 49: Triple Type Graph TG_{B2IT}

The triple graph grammar $TGG_{B2IT} = (TG_{B2IT}, S_{B2IT}, TR_{B2IT})$ specifies how business service models and IT service models given as ABT-Reo diagrams are related and its type graph is shown in Fig. 49 in abstract and concrete syntax. The language of ABT-Reo diagrams is used for both, the source and the target language and the type graph shows a correspondence between the relevant types, which are "ABT" for abstract behaviour type elements and "Point" for the gluing points between input and output ports of ABT elements and Reo connectors. The start graph S_{B2IT} is empty and Figures 50 to 56 show the triple rules of TR_{B2IT}. Each rule specifies a pattern that describes how particular fragments of business and IT models shall be related.

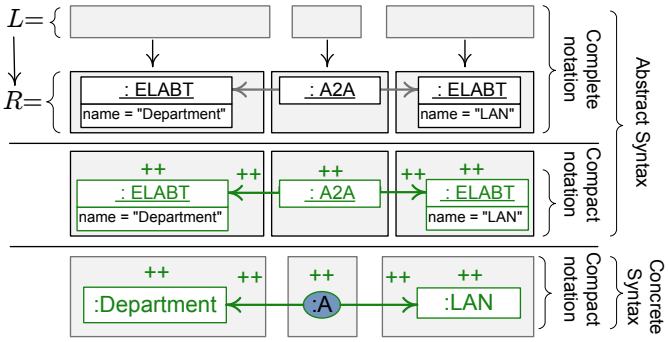


Figure 50: Triple Rule "DepartmentToLAN"

The first rule "DepartmentToLAN" synchronously creates two ABT elements and a correspondence node that relates them. This reflects the general correspondence between departments in the business view and the installed local area networks in the IT view. The rule is presented in

complete and in compact notation as well as using the concrete syntax. The compact notation combines the left and the right hand side of a rule. All elements that are created by the rule, i.e. which appear in the right hand side only, are marked by green line colour and double plus signs. The concrete syntax shows the ABT diagrams in visual notation, where the attribute "name" is used as label of the visual elements.

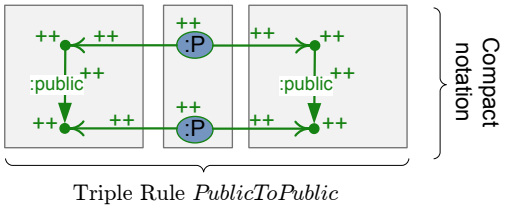


Figure 51: Triple Rule "PublicToPublic"

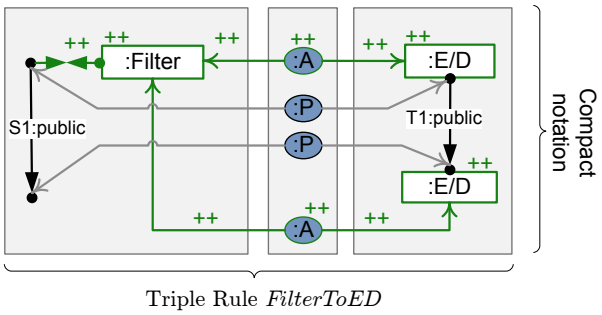


Figure 52: Triple Rule "FilterToED"

The further figures show rules in compact notation and in concrete syntax. Similarly to the first rule the rule "PublicToPublic" has also an empty left hand side. It synchronously creates two Reo connectors on both sides and they are related by the points at their input and output ports. The rule "FilterToED" is slightly more complex and shows the pattern how filters in business models correspond to encrypted connections in the related IT model. This reflects the abstract business requirement of hiding confidential information and its possible implementation by encryption in the IT domain. Note that the left hand side of this rule corresponds to the right hand side of rule "PublicToPublic".

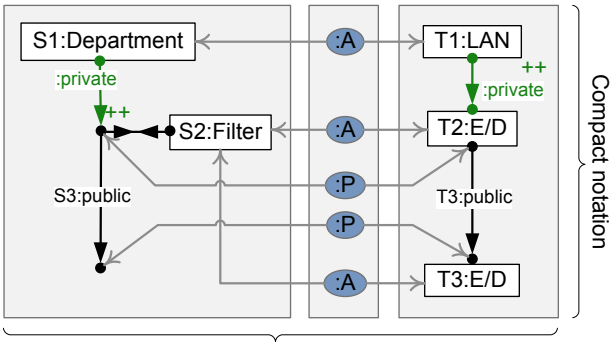


Figure 53: Triple Rule "PrivateInToPrivateIn"

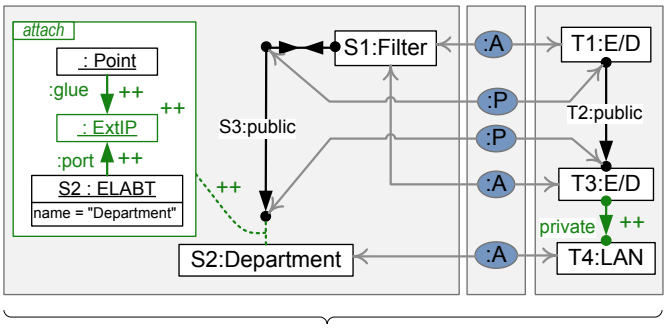


Figure 54: Triple Rule "FilteredOutToPrivateOut"

Private connections leading to related and secured public connections are related by the rule "PrivateInToPrivateIn". The last rule "FilteredOutToPrivateOut" in Fig. **??** specifies how outgoing communication from a secured connection is handled. The private outgoing connection in an IT model corresponds to the gluing of the filtered public connection to the target ABT element, which is defined by the box with the label "attach". This box specifies the explicit creation of elements in the source component based on the underlying abstract syntax, where an input port for the ABT node "S2" and the linking edges to the existing nodes are created.

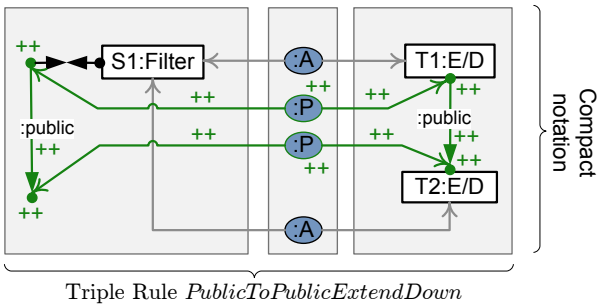


Figure 55: Triple Rule "PublicToPublicExtendDown"

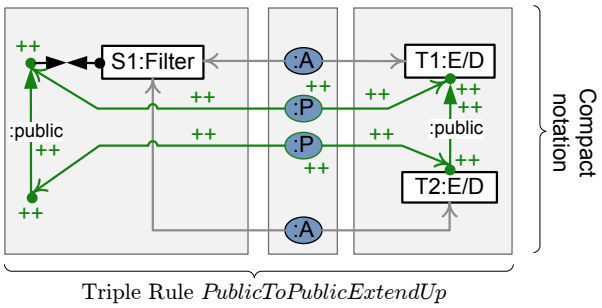


Figure 56: Triple Rule "PublicToPublicExtendUp"

The triple rule "PublicToPublicExtend" specifies the extension for symmetric communication and it is similar to "FilterToED".

Considering the model framework in Fig. 5 there may be the following situations during the development of the models. First of all, two models that should be integrated may be unrelated, thus performing model integration may detect conflicts between them. Furthermore, some model

instances within the model framework may not exist already, e.g. business process models and IT service models usually exist while business service models may not be developed. The interesting challenge is to automatically retrieve parts of the missing models from the existing ones in order to improve interoperability between system components and enterprise components in general. For this purpose, model transformation can be applied e.g. on business process models to derive IT process models and on IT service models to derive basic business service models. The results can be checked against integration conflicts with respect to the other existing models.

The following sections show that triple graph grammars are a suitable basis for the described needs. We exemplarily show how operational rules for model transformation and integration are derived from the original triple graph grammar. The underlying formal construction enables an automatic derivation of the operational rules as described in [27, 7, 32]. The application of the operational rules is controlled, such that correctness and completeness with respect to the patterns are ensured for the resulting models [31]. The techniques are illustrated based on the given scenario of IT and business service models given by ABT-Reo diagrams.

4.2 Model Transformation based on Forward Rules

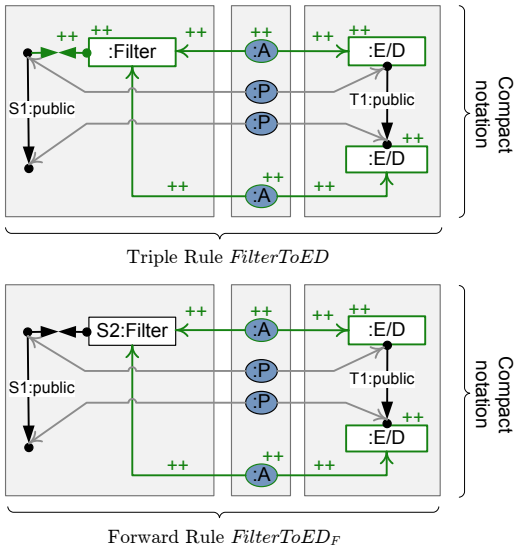
As described in Sec. 4.1 triple rules can be used to specify how two models can be created simultaneously. Thus, triple rules allow the modeler to define patterns of correspondences between model fragments. Based on these triple rules the operational forward rules for model transformations from models of the source language to models of the target language are derived automatically. Since triple rules have a symmetric character, the backward rules for backward model transformations from models of the target to models of the source language are also derived automatically. In this section we present both directions and show the application in our scenario for the forward case. In Sec. 4.4 we will reuse the operational rules for propagating constraints from one domain to a related domain and illustrate the constructions for the backward case.

Operational rules for model transformations are forward and backward rules for forward and backward transformations. Both kinds are derived from the triple rules that specify pattern by pattern how integrated models are created, i.e. how source and target models are developed synchronously. Given a triple rule its forward rule is derived by replacing the source component in the left hand side by the source component in the right hand side. This way the rule requires a complete fragment in the source component of an integrated model and completes the missing parts for the correspondence and target components. Similar to the forward case backward rules are derived in order to perform backward model transformations.

57 shows the triple rule "*FilterToED*" and its derived forward rule "$FilterToED_F$", where the source component is now identical in the left and right hand side of the forward rule. The forward rule is used to transform a filter into two ABT nodes of the type "E/D", which implement the encryption and decryption of communication data. The underlying idea here is that confidential communication in a business universe is filtered out while in the IT universe the data is encrypted for public channels. Since the left hand side of the forward rule contains already all source elements of the right hand side of the triple rule, the item "S1" appears in both, in the left and in the right hand side. The specification of the triple rule and the automatic derivation of its forward rule in the AGT Mathematica (AGT_M) implementation [6] is shown in Fig. 57.

In order to perform model transformations based on the derived forward rules a given source model is extended to a triple graph G_0 with an empty correspondence and an empty target component. The transformation starts with this triple graph. Each step of the transformation starts with the computation of the possible matches from the left hand sides of the forward rules to the current triple graph. A valid match according to the on-the-fly construction in [31] is chosen and the forward step is performed. The resulting target model of a model transformation based on forward rules is obtained by restricting the final triple graph to its target component. This construction leads to a source consistent forward sequence that defines the model transformation sequence.

Example 11 (Model Transformation)**.** *Using the presented triple rules and its derived forward*




```
1 (* creation of graphs SL,CL,TL,SR,CR and TR *)
2 ...
3 FilterToED$L = TGGmakeGraph[SL,CL,TL];
4 FilterToED$R = TGGmakeGraph[SR,CR,TR];
5 (* TGG rule: L, R and application conditions *)
6 FilterToED   = TGGmakeRule[FilterToED$L,FilterToED$R,
7                {}];
8 FilterToEDF  = TGGforwardRule[FilterToED];
```

Figure 57: Triple Rule, Derived Forward Rule and Mathematica Source Code

rules, the business service model $M_{S,B,M}$ is transformed to the IT service model $M_{S,I,M}$. The model transformation sequence consists of the following 6 forward transformation steps and is shown in Figures 58 and 59:

$G_0 \overset{DepartmentToLAN_F}{\Longrightarrow} G_1 \overset{DepartmentToLAN_F}{\Longrightarrow} G_2$
$\overset{PublicToPublic_F}{\Longrightarrow} G_3 \overset{FilterToED_F}{\Longrightarrow} G_4$
$\overset{PrivateInToPrivateIn_F}{\Longrightarrow} G_5 \overset{FilteredOutToPrivateOut_F}{\Longrightarrow} G_6$ *where* $G_0 = (M_{S,B,M} \leftarrow \emptyset \rightarrow \emptyset)$. *At each step a part of the source model is completed by the missing elements in the correspondence and target component. The matches of the forward rules do not overlap on their effective elements, which are given by $R_S \setminus L_S$ of the specified triple rule and visualized by green colour and plus signs in the source components of the triple rules. Furthermore, each element in $M_{S,B,M}$ is matched by an effective element of a forward rules. Both properties are ensured by a formal condition, called source consistency, which controls the forward transformation. This way each source element is translated exactly once and the resulting triple graph is a well formed integrated model. The resulting triple graph G_6 is shown at the bottom of Fig. 59 and coincides almost with the triple graph in Fig. 48. Only the labels "NW4" and "NW7" for the nodes of type "LAN" are left blank. Such information cannot be determined by the information of any source model and therefore, the triple rules create blank labels. But note that real attributes are processed and computed during the model transformation, e.g. the connector type "public" is specified as an attribute of a Reo connector in the abstract syntax. The result of of the forward transformation is given by the target component $M_{S,I,M}$ of $G_6 = (M_{S,B,M} \leftarrow G_{6,C} \rightarrow M_{S,I,M})$. Figure 60 shows the model transformation in one step from its source model as input to its target model as output. The figure additionally contains the corresponding operation calls in the Mathematica implementation AGT_M. Note that some transformation steps in this sequence are sequentially independent, e.g. the second*

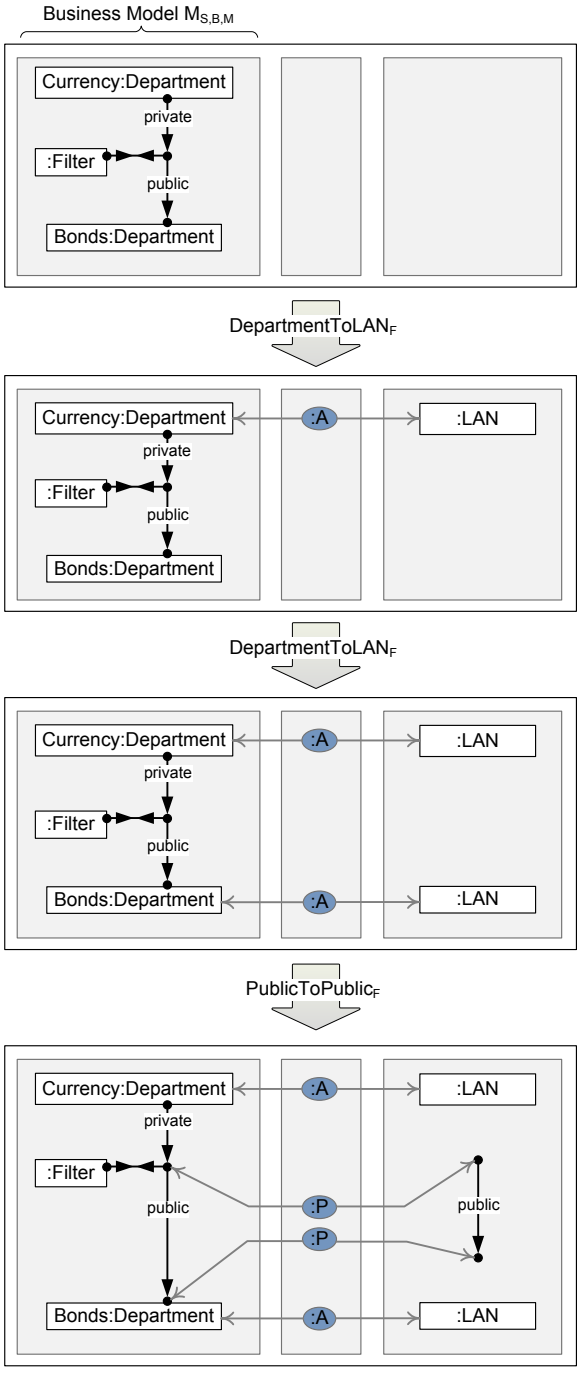


Figure 58: Model Transformation Sequence Part 1

and the third step are independent. They can be switched leading to an equivalent sequence. For this reason, the notion of parallel independence of forward transformation steps in [31] is used and analyzed during the transformation. This allows to avoid the computation of the equivalent sequences.

Model transformations based on source consistent forward sequences are correct and complete with respect to the triple patterns [31], i.e. with respect to the language $VL = \{G \mid \emptyset \Rightarrow^* G \text{ in } TGG\}$ containing the integrated models generated by the triple rules. More precisely, each model

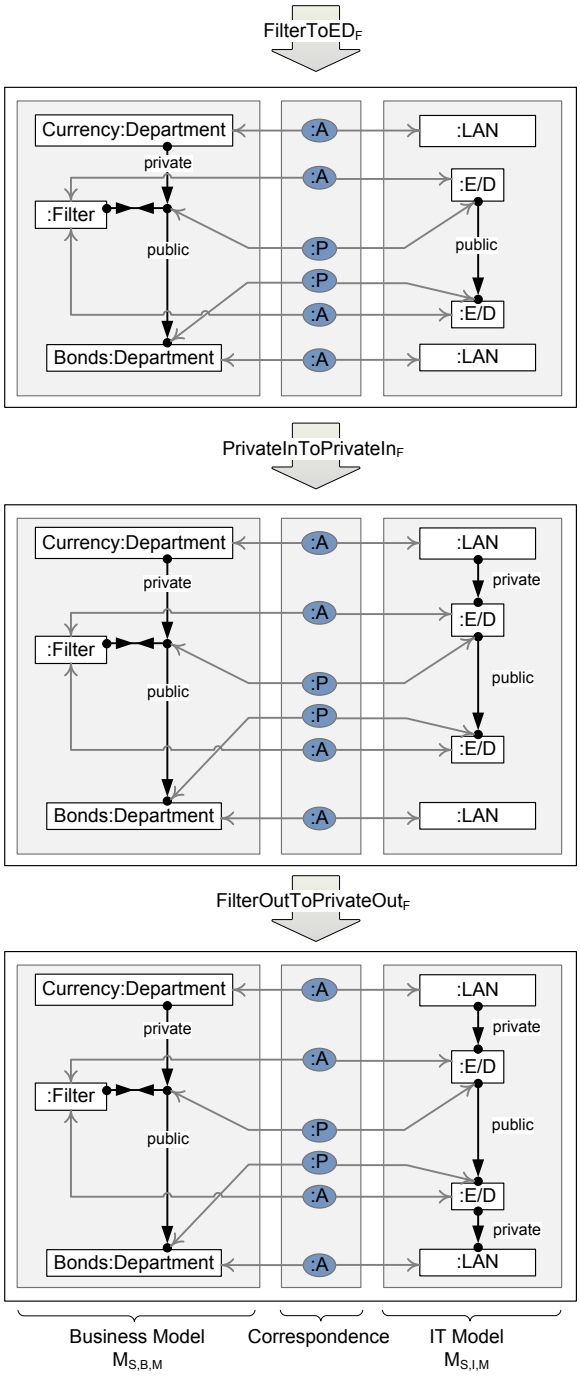
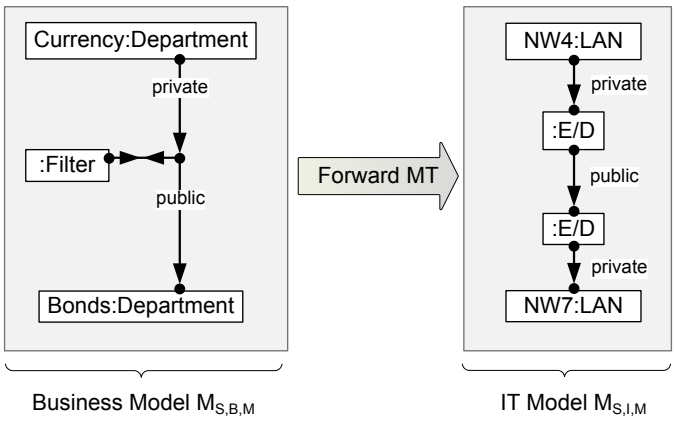


Figure 59: Model Transformation Sequence Part 2

transformation translates a source model into a target model, such that the integrated model that contains both models can be created by applications of the triple rules to the empty start graph. This means that both models can be synchronously created according to the triple patterns. Vice versa, model transformation can be performed on each source model that is part of an integrated model in the generated triple language VL.

The languages of translatable source models VL_S and of reachable target models VL_T are given by




```
1 (* Grammar: Forward Triple Rules ABTReo$RulesMT and
2    Integrated Type Graph TripleTG$A2A *)
3 ABTReo$GrammarMT={ABTReo$RulesMT, TripleTG$A2A};
4 (* Apply Model Transforamation to Model M_SBM *)
5 ModelSIM = TGGmodelTrafo[ModelSBM, ABTReo$GrammarMT];
```

Figure 60: Model Transformation of Service Models

$VL_S = \{G_S \mid (G_S \leftarrow G_C \rightarrow G_T) \in VL\}$ and
$VL_T = \{G_T \mid (G_S \leftarrow G_C \rightarrow G_T) \in VL\}$. Based on these definitions there is the correctness and completeness result below according to Theorems 2 and 3 in [33].

Theorem 4 (Correctness and Completeness)**.**

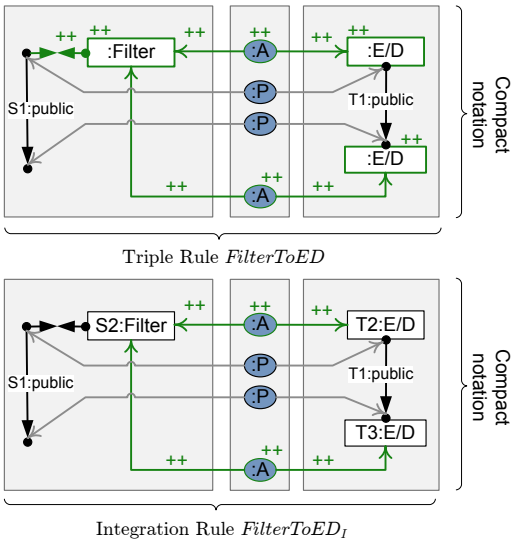
- Correctness: *Each model transformation sequence given by* $(G_S, G_0 \overset{tr_F^*}{\Longrightarrow} G_n, G_T)$*, which is based on a source consistent forward transformation sequence* $G_0 \overset{tr_F^*}{\Longrightarrow} G_n$ *with* $G_0 = (G_S \leftarrow \emptyset \rightarrow \emptyset)$ *and* $G_n = (G_S \leftarrow G_C \rightarrow G_T)$ *is correct, i.e.* $G_S \in VL_S$ *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 sequence* $(G_S, G_0 \overset{tr_F^*}{\Longrightarrow} G_n, G_T)$ *where* $G_0 \overset{tr_F^*}{\Longrightarrow} G_n$ *is source consistent with* $G_0 = (G_S \leftarrow \emptyset \rightarrow \emptyset)$ *and* $G_n = (G_S \leftarrow G_C \rightarrow G_T)$*.*

A comparable result is not available for plain graph transformation approaches, where only syntactical correctness of resulting target models can be analysed. In addition to the correctness and completeness result, the triple graph transformation approach benefits from its intuitive triple patterns of corresponding fragments, which substantially increases usability and maintainability.

4.3 Model Integration based on Integration Rules

The purposes of model integration are first of all interoperability in general and the analysis of consistency within the overall enterprise model in particular. Conceptually, the challenge of model integration is different from model transformation. However, both techniques can be based on triple graph transformation and thus, they are strongly related in our case. The starting set of triple rules is even the same - the only differences occur in the derivation of operational rules and the definition of the control conditions for the execution. This section shows the technical constructions and presents the correctness and completeness result, which is similar to the case of model transformation.

Analogously to forward rules, integration rules are derived from the set of triple rules, which describe the patterns of the relations between two models. An integration rule is obtained from a triple rule by replacing the source and the target component of the left hand side by the source and




```
1 FilterToEDI = TGGintegrationRule[FilterToED];
```

Figure 61: Derived Integration Rule $FilterToED_I$ and Mathematica Source Code

target components of the right hand side, such that only the correspondence part remains different. This way, a match of the rule requires that all fragments of the triple pattern in the source and target component are present before applying the rule, such that only the correspondences are added to complete the triple pattern.

Example 12 (Integration Rule)**.** *Figure 61 shows the triple rule "FilterToED" and its derived integration rule "$FilterToED_I$". The triple rule synchronously extends a public connector by a filter in the business model and its corresponding two de/encryption elements in the IT model. Thus, an application of the integration rule completes the correspondence structure of existing and already related public connectors that have adjacent filter resp. de/encryption elements.*

Model integration based on triple graph transformation is defined by integration sequences, in which the derived model integration rules are applied. Given a source and a target model their integration is performed by completing the correspondence structure that relates both models. The consistency of a model integration is ensured by a formal condition, called S-T-consistency [30]. This condition ensures that the given source and target models are completely parsed using the inverted triple rules restricted to the source and target component, respectively. Thus, each fragment of the models is processed and integrated exactly once.

Example 13 (Model Integration)**.** *Figures 62 to 64 shows the integration of models $M_{S,B,M}$ and $M_{S,I,M}$, i.e. the creation of the correspondence relation between them via a correspondence graph. The model integration is based on an S-T-consistent integration sequence consisting of the following 6 steps using the derived model integration rules as shown in Figures 62 and 63:*

$G_0 \overset{DepartmentToLAN_I}{\Longrightarrow} G_1 \overset{DepartmentToLAN_I}{\Longrightarrow} G_2$
$\overset{PublicToPublic_I}{\Longrightarrow} G_3 \overset{FilterToED_I}{\Longrightarrow} G_4$
$\overset{PrivateInToPrivateIn_I}{\Longrightarrow} G_5 \overset{FilteredOutToPrivateOut_I}{\Longrightarrow} G_6$ *with* $G_0 = (M_{S,B,M} \leftarrow \emptyset \rightarrow M_{S,I,M})$.

In the first four steps some fragments of the source and target components are completed by the missing elements in the correspondence component. In the two last steps no correspondence nodes are created. The reason is the following. The triple rules "PrivateInToPrivateIn" and "FilteredOutToPrivateOut" extend integrated fragments in the source and target model by connecting Reo elements and they do not create any correspondence node. Therefore, the derived

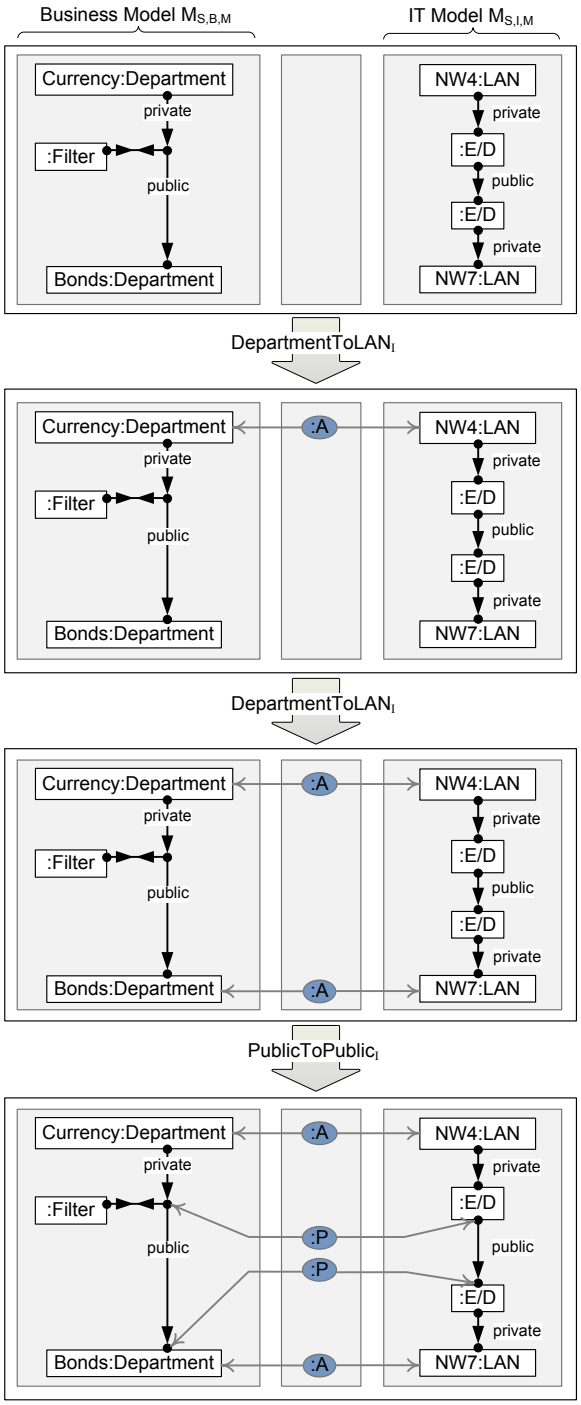


Figure 62: Model Integration Sequence Part 1

integration rules do not create correspondences either. But these integration rules are necessary to ensure correct integrations in the way that the positions of the corresponding private Reo connectors are checked.

The matches of the integration rules do not overlap on their effective elements, which are given by $R_S \cup R_T \setminus L_S \cup L_T$ *of the specified triple rule and visualized by green colour and plus signs in the source and target components of the triple rules. Furthermore, each element in* $M_{S,B,M}$

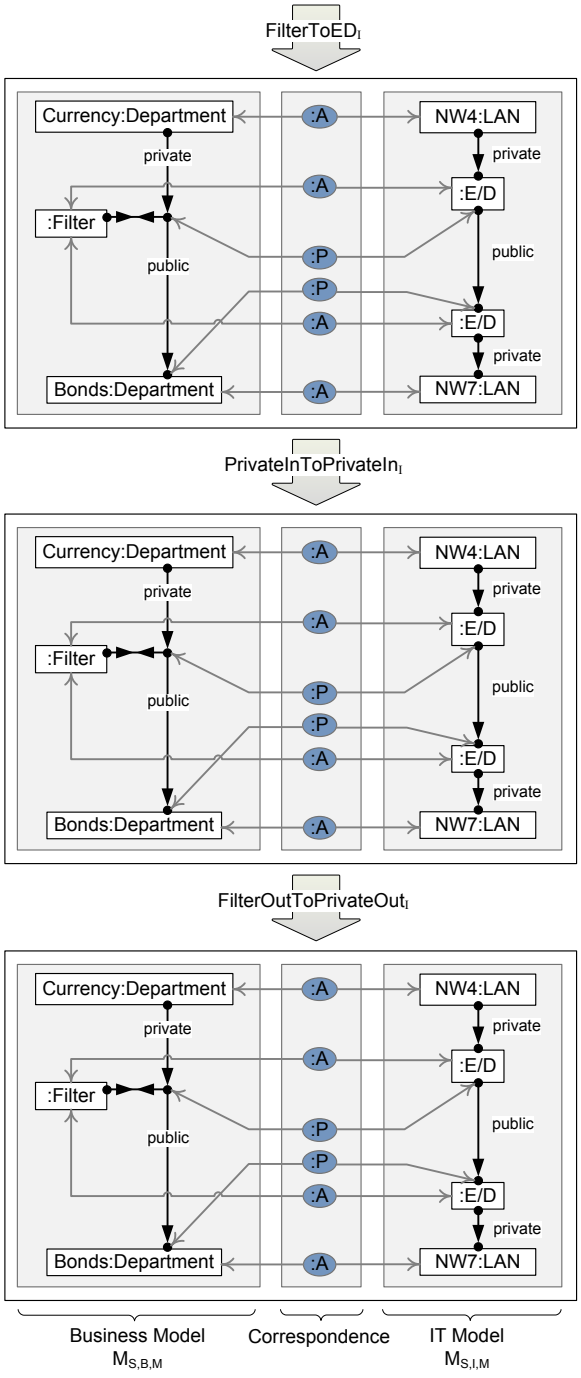


Figure 63: Model Integration Sequence Part 2

and $M_{S,I,M}$ is matched by an effective element of an integration rule. Both properties are ensured by the formal S-T-consistency condition, which controls the integration sequence. This way each source element and each target element is integrated exactly once and the resulting triple graph is a well formed integrated model. The resulting triple graph G_6 is shown at the bottom of Fig. 63 and coincides with the triple graph in Fig. 48.

The result of the model integration is the completely integrated triple graph of the sequence and

it is shown in the bottom part of Fig. 64, where the model integration is presented in a single step from the pair of the source and the model as input to the integrated model as output. Figure 65 shows the corresponding operation calls in the Mathematica implementation AGT_M.

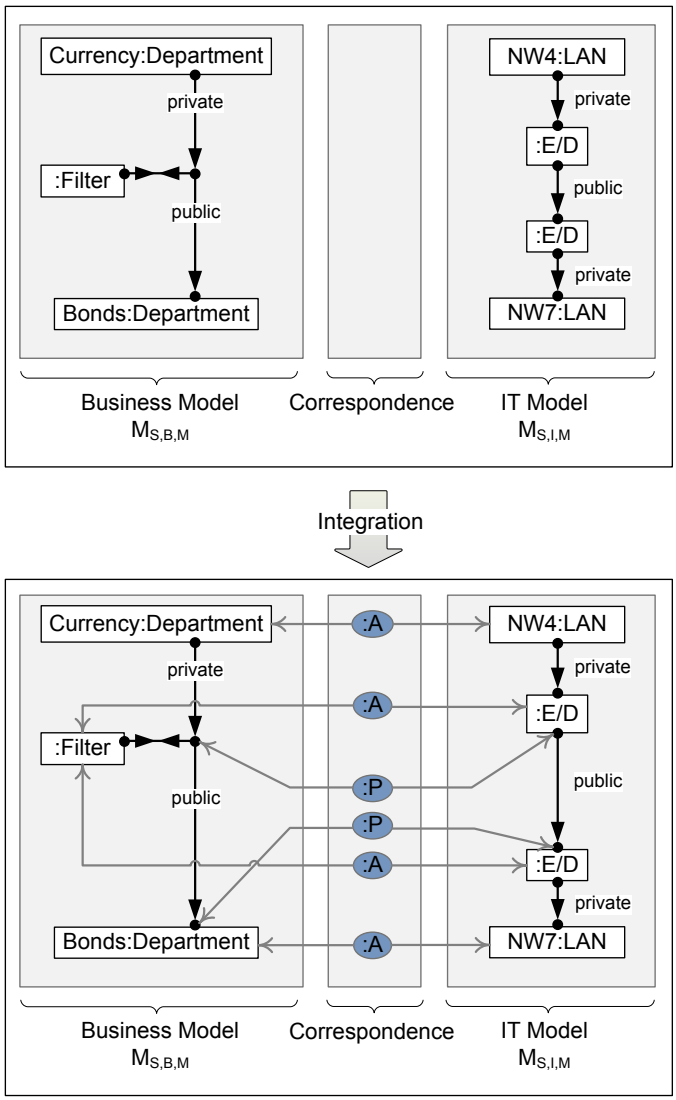


Figure 64: Model Integration for Service Models

```
1 (* Grammar: Integration Triple Rules ABTReo$RulesMI and
2    Integrated Type Graph TripleTG$A2A *)
3 ABTReo$GrammarMI={ABTReo$RulesMI, TripleTG$A2A};
4 (* Apply Model Integration to Model M_SBM *)
5 ModelMI = TGGintegrate[ModelSBM,ModelSIM,ABTReo$GrammarMI];
```

Figure 65: Operation Calls for Model Integration

Integration rules are used to establish or update the correspondences between two models. The model framework in Fig. 5 shows many coordinates, where models should be integrated. Two models are consistent with each other, if they can be completely integrated, otherwise they show

conflicts. Consistency between the models is ensured by checking S-T-consistency [30], which means that the integration has to conform to a parsing of the existing source and target model.

If two models cannot be fully integrated, i.e. no $S-T$-consistent integration sequence can be found, then some parts of the models cannot be related. For instance the IT model may contain some communication paths that are not modelled in the business model. Those synchronization fragments are detected by computing the integration sequences with maximal coverage. Thereafter, the remaining parts are marked for further synchronization that shall be performed by the modelers. The synchronization by the modeler can be supported by the application of the derived forward and backward triple rules in the way that additional fragments in one model are translated to new ones in the other model.

Analogous to the forward case model integration based on triple rules is correct and complete according to Thm. 3 in [30] combined with the the result for the forward case. This means that each model integration leads to a triple graph that can be created by the application of the specified triple rules. Vice versa, integration can be performed for each pair of a source and a target model that can be obtained from a model in the triple language VL generated by the triple rules.

Theorem 5 (Correctness and Completeness).

- Correctness: *Each model integration sequence given by* $((G_S, G_T), G_0 \overset{tr_I^*}{\Longrightarrow} G_n, G_n)$ *which is based on the S-T-consistent integration sequence* $G_0 \overset{tr_I^*}{\Longrightarrow} G_n$ *with* $G_0 = (G_S \leftarrow \emptyset \rightarrow G_T)$ *is correct, i.e.* $G_n \in VL$.

- Completeness: *For each* $G_n \in VL$ *there exists a model integration sequence* $((G_S, G_T), G_0 \overset{tr_I^*}{\Longrightarrow} G_n, G_n)$, *such that* $G_0 \overset{tr_I^*}{\Longrightarrow} G_n$ *is S-T-consistent,* $G_0 = (G_S \leftarrow \emptyset \rightarrow G_T)$ *and* $G_n = (G_S \leftarrow G_C \rightarrow G_T)$.

After we have shown how models can be integrated by model integration based on triple graph grammars, the next section focusses on a further aspect concerning the analysis of the requirements between different domains.

4.4 Rule based Propagation of Constraints

Each domain in the model framework for enterprise modelling in Fig. 5 is faced with several non-functional requirements and quite a lot of them are hard requirements given by e.g. security rules that the enterprise has to implement. This means that the enterprise models have to be checked against the non-functional requirements. A local analysis of these requirements for single models is presented in Sec. 3.3 based on intuitive and visual graph constraints that are checked against the abstract syntax graph of the model.

However, even if the existing models of one dimension in the enterprise model framework in Fig. 5 respect the non-functional requirements, there may be some other models in an interconnected dimension that implicitly violate the requirements. This effect occurs if there are e.g. IT service models that are not related with business models, because the business models for this aspect of the enterprise are currently not developed. In this case, the implementation of the modelled IT structure may lead to a system, in which the non-functional business requirements are not respected. The solution is to transfer the requirements from the specific domain to its interconnected domains and to analyze the propagated requirements on the models of the new domains.

In this section, we show how the propagation of graph constraints from one domain to an interconnected other domain can be performed, such that the constraint can be interpreted in the new domain. This new technique is based on the triple patterns for model transformation and model integration, which we used already in Sections 4.2 and 4.3. Since graph constraints in general consist of model fragments only, instead of full models, we have to adapt the model transformation technique, such that the model transformation rules can be applied on fragments. The left hand sides of the model transformation rules may be to large for the small fragments. For this reason, we perform a maximal partial matching and borrow the missing parts in order to

apply the rule. This way, the source component is extended by the borrowed elements, which was not the case for the model transformations in Sec. 4.2.

Figure 66: Graph Constraint for IT Models

In order to illustrate the mechanism of constraint propagation we reconsider the graph constraint for IT models in Sec. 3.3 for ensuring secure communication over public channels in local are networks, which is shown in Fig. 66 in concrete syntax. The non-functional requirement is formalized by the graph constraint in the way that each public Reo connector that can be found in a model (premise graph P) has to be connected to ABT elements with encryption/decryption functionality (conclusion graph C).

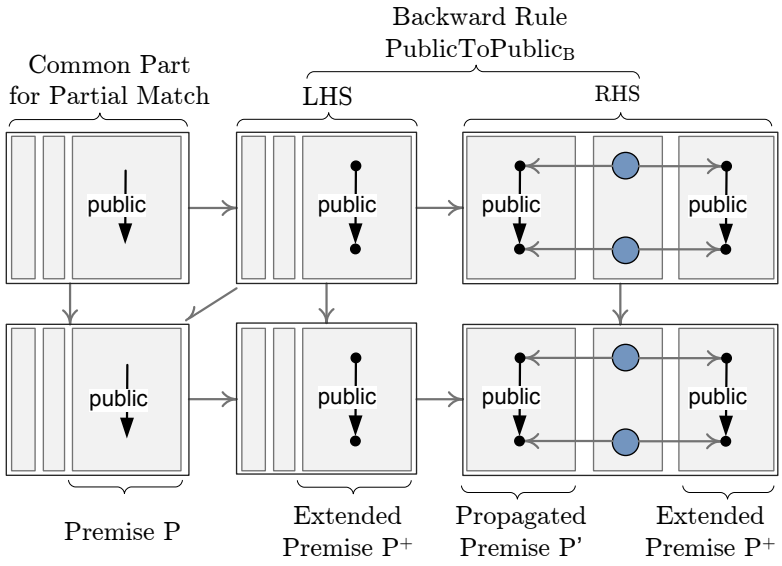


Figure 67: Propagation with Borrowing for Premise P

Example 14 (Propagation of Graph Constraint)**.** *The graph constraint "publicIsEncrypted" in Fig. 66 for IT service models is propagated to a graph constraint for business models. This means that we perform two backward transformations with borrowing - one for the premise graph of the constraint and one for the conclusion graph. Borrowing means that we allow partial matching for the model transformation rules and add those parts, which are missing for deriving a total match [6]. The borrowing is performed by a gluing based on a pushout along the common parts of the left hand side of a rule and the triple graph on which the rule is applied. This construction is illustrated in Fig. 67 for the premise graph of "publicIsEncrypted". The rule "PublicToPublic$_B$" requires a Reo connector with ports and points in the left hand side of the rule and therefore, the missing ports and points are borrowed for the premise P leading to the extended premise graph P^+, which in this example coincides with the left hand side of the rule. The application of the rule results in a triple graph containing the extended premise P^+ and the propagated premise P', which coincide in this example as shown in Fig. 67. This step leads to the first step of the propagation of the conclusion C, which is shown in Fig. 68. In the second step of the propagation of the conclusion graph C we can apply the rule "FilterToED$_B$" and derive the triple graph containing the original conclusion C and the the propagated conclusion C', which is shown in Fig. 68. The propagation leads to the constraint "publicIsFiltered" in Fig. 69. A check of this new constraint*

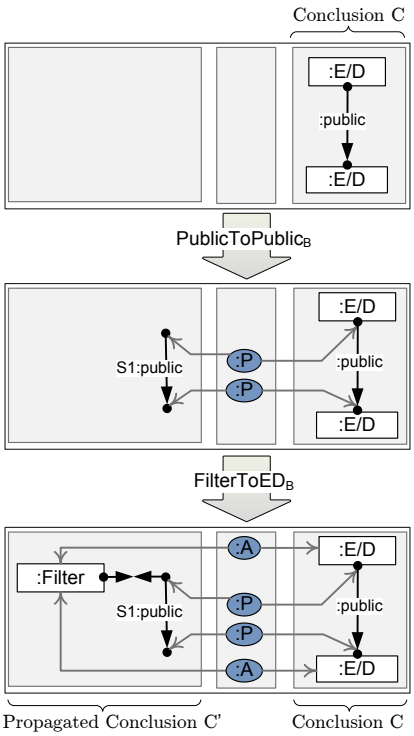


Figure 68: Propagation for Conclusion C

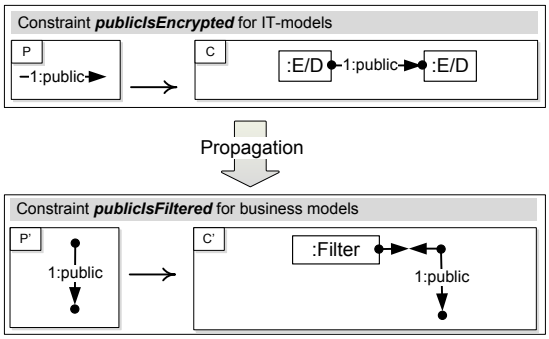


Figure 69: Constraint Propagation

against the model $M_{S,B,M}$ in Fig. 63 shows that the constraint is fulfilled, which can be interpreted as consistency with respect to the security requirement for IT models.

4.5 Summary of Achievements for Inter-Model Techniques

The described and illustrated techniques for model transformation, integration and constraint propagation in this section improve the interoperability between the business and the IT service models given by ABT-Reo diagrams. The techniques are general, such that they should be applicable to the other visual languages and coordinates in the model framework for enterprise modelling in Fig. 5 as well. The benefits of the techniques can be described as follows. Model transformation enables the construction of model stubs that can be refined by the experts for the specific domain. Model integration establishes the correspondences between the existing models e.g. for data interchange and furthermore, conflicts between models can be detected and highlighted. According to Theorems 4 and 5 model transformation and model integration based on triple graph grammars

are correct and complete. Finally, propagation of graph constraints supports the analysis of models with respect to non-functional requirements that can be relevant for different domains in the model framework.

5 Related Work

The related work covers three parts. The first part addresses requirements about enterprise modelling and enterprise models as they can be found in numerous publications and studies. These requirements are put in relation to the work presented in this paper. The second part focusses on transformation techniques that are used in today's industrial environments. They are compared with the technique of algebraic graph transformation in part three that is used in the context of this study.

5.1 Enterprise Modeling and Enterprise Models

5.1.1 Paradigms, Frameworks, Architectures

There are a wide variety [34] of paradigms, frameworks and architectures for enterprise modelling [35, 36] available. We focus here explicitly on the requirements that emerged in existing approaches and that are relevant for the potential solution.

Enterprise engineering or modelling as it is discussed in the literature takes two points of view. The first is about building organizational models [35]. The second is about IT models that lead to an IT architecture and implementation to automate business processes [37]. Because of the scope of this paper we will focus explicitly on requirements for organizational models that abstracts away IT implementation details. We believe that this will reduce the overall complexity and avoid the burden of building and maintaining fat models [38]. The reason for such problems is often an insufficient knowledge in the field about available types of models and potential modelling techniques [39]. The types or models required to handle organizational phenomena need to be expressive enough to describe the static and dynamic aspects of an organization. They need to be open for extensions as well as being formal to enable fully automated verification and validation [40]. Even so enterprise models are created as descriptive models they should support organizational control and automation by tools later on which implies that an organizational model will configure an IT platform running business processes [40]. However, it should not cover IT implementation aspects [40].

Our new model framework has a good potential to address these requirements. The modelling framework organizes the interplay of lean and simple models that can be integrated towards an holistic view. The sound integration of human-centric and machine-centric models supports current modelling workflows in the decentralized organization of Credit Suisse and enables model-checking capabilities. By putting the focus on services, processes and rules such an organizational model can serve as a configuration for service oriented IT architectures. It is further able to capture the understanding of the domain which is already in use.

5.1.2 Enterprise Modelling aspects

The purpose of enterprise modelling is to understand [41], develop [42], manage, optimize [42] and control [43], [44] an organization.

From a social point of view, we can assume that organizational knowledge is usually distributed and that the people who carry organizational knowledge are limited by their bounded rationality [45]. Therefore, building small and focussed models that can be locally integrated should be given preference over creating big and extensive models that are integrated by a global meta-model [45]. Because people are considered to be the crucial source of organizational knowledge their conceptual understanding needs to be smoothly combined with formal methods regarding the models, the modelling process and modelling techniques [46]. Formal methods used in this

context need to support decentralized organizational workflows and agile modelling as well as collaborative modelling [47].

From a domain point of view, enterprise models are class and instance models because they need to represent the organizational situation as it is [48].

From a technical point of view, domain specific modelling environments are needed that extend the facility of available meta-CASE tools by giving priority to model transformation and integration. Such environments must support the evolution of domain languages as well as their artefacts [49].

Finally, because modelling causes significant costs [50] reuse of models, modelling elements and modelling procedures is required [51]. So, a catalogue of modelling elements and modelling procedures can help here [52]. Such a catalogue will contain objects that are reconstructed given environmental realities and objects that are normatively developed based on available organizational theories. Quality assurance of catalogue elements is assumed to be highly effective when reuse scales up [53].

Our new model framework has the potential to cover these requirements. Lean and focussed models allow on one hand decentralized and agile modelling, but on the other hand there are also integration techniques to take care of a holistic view. Instance- and class modelling is possible likewise. The declarative nature of algebraic graph transformation makes the formalism highly usable. Negative application conditions and graph constraints allow to control the modelling process. Diagram-like models and text-like models can be handled homogeneously by using their abstract syntax. The interplay of human- and machine-centric models joins flexibility requirements during the modelling process with formal requirements during model evaluation. By using catalogs of model elements and assemblies a lego-like modelling system could be provided. The same applies to transformation rules or sets of transformation rules. Algebraic graph transformation shows good potential to help to address all this in a formal way which will support also automation and quality assurance.

5.1.3 Alignment aspects

As already stated in section 2.4 and in [13] and [14] alignment aspects between business and the IT models are of high importance. The presented approach shows potential to support such a flexible alignment in a (semi)- automated fashion, top-down as well as bottom-up, by the help of intra-model integration techniques.

5.1.4 Adaptive Modelling

Adaptive modelling [54], [55] shows up when domain languages and their artefacts evolve [56]. This happens because domain knowledge is usually given in an implicit way and can be made explicit only during the modelling process itself. Another reason why this happens is because the environment that is modelled changes [57]. Here, new kinds of requirements may require a change of domain languages as well as their already existing artefacts [58]. Adaptive modelling can be supported by techniques of algebraic graph transformation, because the rule based approach allows to handle modifications and adaptions in an easy way.

5.1.5 Simulation and Analysis aspects

As pointed out in [59] and [60] it should be possible to simulate all kinds of enterprise models. Simulation models for enterprise planning purposes can be derived by using model transformation rules. In addition to that, the tool environment AGG [61] can be used for analysis and simulation of graph grammars.

5.2 Model Transformation techniques

A taxonomy of model transformation techniques is presented in [62] and it can be used to help developers deciding which model transformation approach is best suited to deal with a particular

problem. By definition a transformation is the automatic generation of a target model from a source model, according to a transformation definition, that describes how a model or a set of models in the source language can be transformed into a model or a set of models in the target language. Transformations can be endogenous or exogenous. The first case is about transformations of models that share the same language. The second case is about transformations of models that are build on different languages. Further on, a transformation can be horizontal or vertical. A horizontal transformation is a transformation where the source and target model is at the same abstraction level. A vertical transformation has source and target models at different abstraction levels.

Model transformations are based on a concept of models, where one area of models is the model-driven architecture initiative created by the Object Management Group [63]. Here, platform-specific models are generated from platform independent models. The existing model-to-model transformation approaches can be differentiated into direct manipulation approaches, relational approaches, graph-transformation-based approaches, structure-driven approaches and hybrid approaches. The direct manipulation is the most low-level approach. It offers little or no support or guidance in implementing transformations. The relational approach seem to strike a well balance between flexibility and declarative expressions. They provide flexible scheduling and good control of non-determinism. Some of the QVT [64] submissions fit into this category. The graph-transformation-based approaches are powerful and declarative, but sometimes also complex. The complexity stems from the non-determinism in scheduling and application strategy, which requires careful consideration of termination of the transformation process and the rule application ordering (including the property of confluence). There is a large amount of theoretical work and good experience with research prototypes. The structure-driven approach groups pragmatic approaches that were developed in the context of certain kinds of applications and the hybrid approach allows the user to mix and match different concepts and paradigms depending on the application [65].

5.3 Graph Transformation Approaches

Graph transformation has a long tradition [1], a well-founded theory [2, 5] and has been widely used for expressing model transformations [7, 27, 30, 31, 32, 33]. Especially transformations of visual models can be naturally formulated by graph transformations, since graphs are well suited to describe the underlying structures of models [66].

By comparing graph transformation approaches and QVT we see that in all approaches considered, the typing information is given by an attributed type graph or meta-model which contains the structural information, inheritance concepts and multiplicity constraints. AToM3 allows constraints to be expressed in Phython, algebraic graph transformation shows up with graph constraints, in QVT typing information is provided from the source and target meta-models for model transformation. In the standard graph transformation approach [5], the type graph of the model transformation consists of the source type graph, the target type graph and additional reference nodes and edges needed during the model transformation. In the triple graph grammar approach proposed in [7, 27, 30, 31, 32, 33] source and target graphs are connected by a correspondence graph and triple rules can generate automatically forward and backward model transformation rules. Furthermore, there are general results that ensure correctness and completeness of model transformations [31]. We can notice that the graph transformation approaches and QVT share a number of commonalities. The typing concepts by type graphs or the almost equivalent concept of meta-models can be found in all approaches. Moreover, all transformation approaches considered are rule-based, even QVT with its concept of relations between domain models is very close. While the simple rule-based approach is unidirectional, triple graph grammars and QVT relations focus more on bidirectional or even multi-dimensional transformations. All approaches follow an idea of pre- and post-conditions expressed by some patterns, equipped with typical actions changing the models. Main differences can be found in the description of additional attributes using Java, Python, ASM or OCL as languages for attribute computations as well as conditions. Moreover, the control of rule applications ranges from pure rule-based approaches allowing a high degree of non-determinism, to rather controlled rule applications using mainly automata-based descriptions

[66].

Compared with alternative solutions the technique of algebraic graph transformation is build on a body of theory that allows formal proofs and that can therefore guarantee qualities of model transformation and model integration results [30, 31, 32, 33]. It is able to check constraints about models. It is able to demonstrate how constraints can be propagated by the help of triple graph rules to check for inconsistencies with constraints of models of different type.

6 Conclusion: Summary and Future Work

The integration of business and IT models on one hand and human-centric and machine-centric models on the other hand are major challenges in enterprise modelling. In addition to that there are several further requirements depending on the enterprise domain and also specific to the needs of the enterprise. In this paper, we have proposed a new enterprise modelling framework based on algebraic graph transformation techniques that is focussed on the requirements of Credit Suisse with the purpose to improve today's enterprise models and the interoperability between them. Furthermore, we presented suitable techniques that support the development and analysis of the models with respect to security, risk and compliance, which are major needs for Credit Suisse. There is a substantial potential that the results can be applied to enterprises with similar needs as well.

The first main contribution of this paper is the proposal of a new model framework with decentralized models in three different dimensions including intra- and inter-modelling techniques in order to support interoperability between them. Moreover, we have discussed how the requirements for modeling and interoperability in this framework can be supported in general by algebraic graph transformation techniques and tools

The second main contribution of this paper is the aggregation of suitable formal techniques for model development, analysis, transformation, integration and propagation of requirements. We illustrated their application on formalised business and IT service models which are situated in the machine centric modelling dimension of the model framework. They shall be aligned with human centric domain languages as explained in Sec. 2.

In order to support flexible modelling in visual notation we introduced the notion of a construction and a correction grammar (C&C system). This way the editing steps of a model development environment can be equipped with the corresponding construction rules that realise the formal construction of the abstract syntax graphs. The correction of models with respect to the well-formedness rules of the language is performed on the basis of the correction grammar. This grammar enables an automated detection and highlighting of the patterns that violate the well-formedness rules. We further explained how the soundness of the C&C system with respect to the set of well-formed models of the modelling language can be analysed using the results for local confluence and termination according to Theorems 1 and 2, which are central results of algebraic graph transformation systems.

Moreover, we have shown how a substantial part of the functional and non-functional requirements in the domain of the enterprise can be formalized in an intuitive but formal way by graph constraints in order to perform automated checks of the models against the requirements. The validity of graph constraints can be preserved during the development of models if certain conditions for the underlying transformation steps are fulfilled, as stated by Thm. 3.

As third main contribution of this paper, we have shown how interoperability between models of different dimensions of the framework can be achieved in order to support the development process of the models and to improve maintainability and the possibilities for analysis. For this purpose we have presented how model transformation and model integration are used to construct and integrate models in order to relate their components and to detect inconsistencies between them. Both techniques are based on triple graph grammars, where patterns describe how essential model fragments of the source and target model shall be related. The techniques can be applied in an automated way and they are shown to be correct and complete according to Theorems 4 and 5.

Moreover, we have illustrated how the requirements specified by graph constraints for one domain can be propagated to other models in the enterprise framework in order to support consistency of related models.

Altogether, this paper should be considered as a proposal for a new enterprise model at Credit Suisse and similar decentralized organizations. In fact, in addition to the general ideas, we have only shown first steps of how to realize this enterprise model using formal methods and tools in the area of algebraic graph transformations. We mainly have focussed on modelling and interoperability of business and IT service models using ABT-Reo diagrams. It remains for future work to extend this to human-centric models based on other formal specification techniques. Moreover, it remains to show semantical compatibility and to develop techniques for the overall integration of all decentralized models. Last but not least, it is a key challenge to implement the new enterprise model at Credit Suisse.

References

[1] Ehrig, H., Pfender, M., Schneider, H.: Graph grammars: an algebraic approach. In: 14th Annual IEEE Symposium on Switching and Automata Theory, IEEE (1973) 167–180

[2] Rozenberg, G., ed.: Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations. World Scientific (1997)

[3] Bardohl, R., Taentzer, G., Minas, M., Schürr, A.: Application of Graph Transformation to Visual Languages. In Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G., eds.: Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages and Tools, World Scientific (1999)

[4] Baldan, P., Corradini, A., Montanari, U., Rossi, F., Ehrig, H., Löwe, M.: Concurrent Semantics of Algebraic Graph Transformations. In Rozenberg, G., ed.: The Handbook of Graph Grammars and Computing by Graph Transformations, Volume 3: Concurrency, Parallelism and Distribution. World Scientific (1999) 107–188

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

[6] Brandt, C., Hermann, F., Engel, T., Adamek, J., Schölzel, H.: Security and Consistency of IT and Business Models at Credit Suisse realized by Graph Constraints, Transformation and Integration using Algebraic Graph Theory (Long Version). Technical report, Technische Universität Berlin,Fakultät IV (to appear 2009) draft version available: `http://tfs.cs.tu-berlin.de/publikationen/Papers09/BHEE09.pdf`.

[7] Kindler, E., Wagner, R.: Triple Graph Grammars: Concepts, Extensions, Implementations, and Application Scenarios. Technical Report TR-ri-07-284, Universität Paderborn (2007)

[8] 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`.

[9] OMG: Catalog Of OMG Business Strategy, Business Rules And Business Process Management Specifications. (2009) `http://www.omg.org/technology/documents/br_pm_spec_catalog.htm`.

[10] Groote, J.F., Mathijssen, A., Reniers, M., Usenko, Y., van Weerdenburg, M.: The formal specification language mcrl2. In Brinksma, E., Harel, D., Mader, A., Stevens, P., Wieringa, R., eds.: Methods for Modelling Software Systems (MMOSS). Number 06351 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany (2007)

[11] Arbab, F.: Abstract Behavior Types: A Foundation Model for Components and Their Composition. Science of Computer Programming **55** (March 2005) 3–52

[12] Brandt, C., Hermann, F., Engel, T.: 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: Proc. International Workshop on Dynamic and Declarative Business Processes (DDBP 2009), IEEE Xplore Digital Library (2009) (accepted).

[13] Zhang, L.J., Zhang, J., Cai, H.: Enterprise modeling. In: Services Computing, Springer (2007) 259–274

[14] Wegmann, A., Lê, L.S., Regev, G., Wood, B.: Enterprise modeling using the foundation concepts of the rm-odp iso/itu standard. Inf. Syst. E-Business Management **5**(4) (2007) 397–413

[15] Pemmaraju, S., Skiena, S.: Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica. Cambridge University Press, New York (2003) The University of Iowa and SUNY at Stony Brook.

[16] Brewer, D.F.C., Nash, M.J.: The chinese wall security policy. In: IEEE Symposium on Security and Privacy. (1989) 206–214

[17] Klüppelholz, S., Baier, C.: Symbolic model checking for channel-based component connectors. In: Proc. of FOCLASA'06. (2006)

[18] Arbab, F.: Reo: a channel-based coordination model for component composition. Mathematical Structures in Computer Science **14**(3) (2004) 329–366

[19] Wirsing, M., Pattinson, D., Hennicker, R., eds.: Recent Trends in Algebraic Development Techniques, 16th International Workshop, WADT 2002, Frauenchiemsee, Germany, September 24-27, 2002, Revised Selected Papers. In Wirsing, M., Pattinson, D., Hennicker, R., eds.: WADT. Volume 2755 of Lecture Notes in Computer Science., Springer (2003)

[20] Scheer, A.W.: ARIS-Modellierungs-Methoden, Metamodelle, Anwendungen. Springer, Berlin/Heidelberg (2001)

[21] Boehmer, W., Brandt, C., Groote, J.: Evaluation of a business continuity plan using process algebra and modal logic. IEEE TIC Toronto (2009)

[22] Reisig, W.: Petri Nets: An Introduction. Volume 4 of Monographs in Theoretical Computer Science. An EATCS Series. Springer (1985)

[23] van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases **14**(1) (2003) 5–51

[24] Rensink, A.: Representing first-order logic using graphs. In Ehrig, H., Engels, G., Parisi-Presicce, F., Rozenberg, G., eds.: Proc. Int. Conf. on Graph Transformation (ICGT'04). Volume 3256 of Lecture Notes in Computer Science., Springer (2004) 319–335

[25] Jurack, S., Lambers, L., Mehner, K., Taentzer, G., Wierse, G.: Object Flow Definition for Refined Activity Diagrams . In Chechik, M., Wirsing, M., eds.: Proc. Fundamental Approaches to Software Engineering (FASE'09). Volume 5503 of LNCS., Springer (2009) 49–63

[26] Brandt, C., Hermann, F., Engel, T.: 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 Exploring Modeling Methods in Systems Analysis and Design 2009 (EMMSAD'09). Volume 29 of LNBIP., Heidelberg, Springer Verlag (2009) 339–352

[27] Schürr, A.: Specification of Graph Translators with Triple Graph Grammars. In Tinhofer, G., ed.: Proc. of WG'94. Volume 903 of LNCS., Springer (1994) 151–163

[28] Ermel, C., Biermann, E., Ehrig, K., Taentzer, G.: Generating Eclipse Editor Plug-Ins using Tiger. In Schürr, A., Nagl, M., Zündorf, A., eds.: Applications of Graph Transformation with Industrial Relevance, Proceedings of the Third International AGTIVE 2007 Symposium. Volume 5088 of LNCS., Heidelberg, Springer (2008) 583–585

[29] Object Management Group: Meta-Object Facility (MOF), Version 2.0. (2006)

[30] Ehrig, H., Ehrig, K., Hermann, F.: From Model Transformation to Model Integration based on the Algebraic Approach to Triple Graph Grammars. In Ermel, C., de Lara, J., Heckel, R., eds.: Proc. Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT'08). Volume 10., EC-EASST (2008)

[31] Ehrig, H., Ermel, C., Hermann, F., Prange, U.: On-the-Fly Construction, Correctness and Completeness of Model Transformationsbased on Triple Graph Grammars. In Schürr, A., Selic, B., eds.: ACM/IEEE 12th International Conference on Model Driven Engineering Languages and Systems (MODELS'09). Volume 5795 of lncs., Springer (2009) 241–255 To appear.

[32] Ehrig, H., Ehrig, K., Ermel, C., Hermann, F., Taentzer, G.: Information preserving bidirectional model transformations. In Dwyer, M.B., Lopes, A., eds.: Fundamental Approaches to Software Engineering. Volume 4422 of LNCS., Springer (2007) 72–86

[33] Ehrig, H., Hermann, F., Sartorius, C.: Completeness and Correctness of Model Transformations based on Triple Graph Grammars with Negative Application Conditions. In Heckel, R., Boronat, A., eds.: Proc. Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT'09), EC-EASST (2009)

[34] Giaglis, G.M.: A taxonomy of business process modeling and information systems modeling techniques. International Journal of Flexible Manufacturing Systems **13**(2) (2001) 209–228

[35] Sousa, G., Van Aken, E., Rentes, A.: Using enterprise modeling to facilitate knowledge management in organizational transformation efforts. Portland International Conference on Management of Engineering and Technology. IEEE. **1** (2001) 62

[36] Lillehagen, F., Krogstie, J.: State of the art of enterprise modeling. In: Active Knowledge Modeling of Enterprises, Springer (2008) 91–127

[37] Lankhorst, M.: Enterprise Architecture at Work. Modelling, Communication and Analysis. Springer (2005)

[38] Pereira, C.M., Sousa, P.: A method to define an enterprise architecture using the zachman framework. In: SAC '04: Proceedings of the 2004 ACM symposium on Applied computing, New York, NY, USA, ACM (2004) 1366–1371

[39] Camarinha-Matos, L.M., Afsarmanesh, H., Ollus, M., eds.: Virtual Organizations. Springer (2005)

[40] Popova, V., Sharpanskykh, A.: A formal framework for modeling and analysis of organizations. In Ralyté, J., Brinkkemper, S., Henderson-Sellers, B., eds.: Situational Method Engineering. Volume 244 of IFIP., Springer (2007) 343–358

[41] Delen, D., Dalal, N.P., Benjamin, P.C.: Integrated modeling: the key to holistic understanding of the enterprise. Commun. ACM **48**(4) (2005) 107–112

[42] Dalal, N.P., Kamath, M., Kolarik, W.J., Sivaraman, E.: Toward an integrated framework for modeling enterprise processes. Commun. ACM **47**(3) (2004) 83–87

[43] Delen, D., Pratt, D., Kamath, M.: A new paradigm for manufacturing enterprise modeling: reusable, multi-tool modeling. IEEE Simulation Conference. Proceedings. (1996) 985–992

[44] Delen, D., Pratt, D.B., Kamath, M.: A new paradigm for manufacturing enterprise modeling: reusable, multi-tool modeling. In: WSC '96: Proceedings of the 28th conference on Winter simulation, Washington, DC, USA, IEEE Computer Society (1996) 985–992

[45] Kateel, G., Kamath, M., Pratt, D.: An overview of cim enterprise modeling methodologies. IEEE Simulation Conference. Proceedings. (1996) 1000–1007

[46] Hoogervorst, J.A.: Enterprise Governance and Enterprise Engineering. Springer (2009)

[47] Zhiming, C., Jun, Y.: The process conducting and member audit in the distributed enterprise modeling. IEEE Asia-Pacific Services Computing Conference (2008) 416–420

[48] Agarwal, R., Bruno, G., Torchiano, M.: Enterprise modeling using class and instance models. Seventh Asia-Pacific Software Engineering Conference. IEEE. (2000) 336–343

[49] Englebert, V., Heymans, P.: Towards more extensible metacase tools. In: CAiSE. (2007) 454–468

[50] Work, B., Balmforth, A.: Using abstractions to build standardized components for enterprise models. IEEE Software Engineering Standards Symposium. Proceedings. (1993) 154–162

[51] Sarder, M., Ferreira, S., Rogers, J., Liles, D.: A methodology for design ontology modeling. Portland International Center for Management of Engineering and Technology. IEEE. (2007) 1011–1018

[52] Malone, T., Crowston, K., Lee, J., Pentland, B.: Tools for inventing organizations: toward a handbook of organizational processes. Second Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises. IEEE. (1993) 72–82

[53] Fernandes, J.M., Duarte, F.J.: A reference framework for process-oriented software development organizations. Software and System Modeling **4**(1) (2005) 94–105

[54] Himsl, M., Jabornig, D., Leithner, W., Regner, P., Wiesinger, T., Küng, J., Draheim, D.: An iterative process for adaptive meta- and instance modeling. In Wagner, R., Revell, N., Pernul, G., eds.: DEXA. Volume 4653 of Lecture Notes in Computer Science., Springer (2007) 519–528

[55] Whitman, L., Ramachandran, K., Ketkar, V.: A taxonomy of a living model of the enterprise. In: WSC '01: Proceedings of the 33nd conference on Winter simulation, Washington, DC, USA, IEEE Computer Society (2001) 848–855

[56] Valentin, E.C., Verbraeck, A.: Domain specific model constructs in commercial simulation environments. In: WSC '07: Proceedings of the 39th conference on Winter simulation, Piscataway, NJ, USA, IEEE Press (2007) 785–795

[57] Mertins, K., Jochem, R.: Integrated enterprise modeling: method and tool. SIGGROUP Bull. **18**(2) (1997) 63–66

[58] Han, Y., Tai, S., Wikarski, D., eds.: Engineering and Deployment of Cooperative Information Systems, First International Conference, EDCIS 2002, Beijing, China, September 17-20, 2002, Proceedings. In Han, Y., Tai, S., Wikarski, D., eds.: EDCIS. Volume 2480 of Lecture Notes in Computer Science., Springer (2002)

[59] Sadowski, D., Bapat, V.: The arena product family: enterprise modeling solutions. In: WSC '99: Proceedings of the 31st conference on Winter simulation, New York, NY, USA, ACM (1999) 159–166

[60] Kubota, F., Sato, S., Nakano, M.: Enterprise modeling and simulation platform integrating manufacturing system design and supply chain. IEEE International Conference on Systems, Man, and Cybernetics **4** (1999) 511–515

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

[62] Mens, T., Czarnecki, K., Gorp, P.V.: 04101 discussion – a taxonomy of model transformations. In Bezivin, J., Heckel, R., eds.: Language Engineering for Model-Driven Software Development. Number 04101 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany (2005)

[63] Group, O.M.: Mda specifications. `http://www.omg.org/mda/specs.htm` (2009)

[64] 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)

[65] Czarnecki, K., Helsen, S.: Classification of model transformation approaches. In: OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture. (2003)

[66] Taentzer, G., Ehrig, K., Guerra, E., de Lara, J., Lengyel, L., Levendovszky, T., Prange, U., Varró, D., , Varró-Gyapay, S.: Model transformation by graph transformation: A comparative study. In: ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems, Montego Bay, Jamaica (October 2005)