Modeling and Reconfigurating critical Business Processes for the purpose of a Business Continuity Management respecting Security, Risk and Compliance requirements at Credit Suisse using Algebraic Graph Transformation: Long Version

== DRAFT VERSION (in progress...) ==

April 5, 2010

Version 1.4

Christoph Brandt, Frank Hermann and Jan Friso Groote

Bericht-Nr. 2010/_ _
ISSN 1436-9915

Modeling critical Business Processes for Continuity Management respecting Security, Risk and Compliance requirements using Algebraic Graph Transformation

(long version)

Christoph Brandt[1], Frank Hermann[2], and Jan Friso Groote[3]

1) Université du Luxembourg, SECAN-Lab, Campus Kirchberg,
6, rue Richard Coudenhove-Kalergi, 1359 Luxembourg-Kirchberg, EU
christoph.brandt@uni.lu,
WWW home page: http://wiki.uni.lu/secan-lab

2) Technische Universität Berlin, Fakultät IV,
Theoretische Informatik/Formale Spezifikation, Sekr. FR 6-1,
Franklinstr. 28/29, 10587 Berlin, EU
frank@cs.tu-berlin.de,
WWW home page: http://www.tfs.tu-berlin.de

3) Eindhoven University of Technology, Systems Engineering,
Department of Computer Science. Hoofdgebouw kamer 6.75,
Den Dolech 2, P.O. Box 513, 5600 MB Eindhoven The Netherlands, EU
J.F.Groote@tue.nl,
WWW home page: http://www.win.tue.nl/~jfg

Abstract

Critical business processes can fail. A Business Continuity Management System is a special management system that will define how to recover from such failures and specifies temporary work-arounds to make sure a company is not going out of business in the worst case. However, because today's implementations are primarily organizational best-practice solutions, their security, risk and compliance issues in such a recovery situation are mostly unknown.

Algebraic graph theory can be used as a formal method supporting employees when running business processes need to be reconfigured to recover from specific failures. The example discussed is a loan granting process in a real-world banking environment. Because such a process has to respect certain laws, regulations and rules even in emergency situations, we sketch how this can be done during the process reconfiguration by looking at security, risk and compliance issues, compatible with the graph technique. Furthermore, we show how the analysis can be extended to requirements concerning the information flow using the process algebra *mCRL2*.

Keywords: business continuity management, algebraic graph theory, event-driven process chains, security, risk, compliance, process algebra

1 Introduction

The problem statement can best be described by the empirical study of Knight and Pretty [1]. They show that companies that implemented a Business Continuity Management System (BCMS) are in a better position to survive a disaster that interrupts one of their critical business processes. However, a company's chance to survive is not guaranteed. Sometimes companies fail to recover from a disaster even so they have implemented a BCMS. In the highly regulated environment of banks certain laws, regulations and rules need to be respected as a side constraint even in such a situation which is, to our knowledge, not solved by today.

The research question derived from this situation is about how an effective and efficient BCMS can be put in place that fullfills security, risk and compliance issues derived from the laws, regulations and rules. This question is discussed based on a loan granting process running in the real-world banking environment at Credit Suisse (CS). The challenge is to generate all continuity processes to a given critical business process and its continuity fragments, such that security, risk and compliance side-constraints are respected. The purpose is to enable an optimal choice of optimal continuity processes and to enable case-based decisions.

This paper presents contributions in the area of business continuity management (BCM) with respect to security, risk and compliance and in the area of algebraic graph transformation (AGT). Given a declarative process model and continuity snippets all possible continuity processes that respect given side constraints can be generated. So, for all combinations of modeled failures it is possible to check if sound continuity processes are available. Therefore, it can be tested beforehand if a BCMS is complete as a whole. In doing so, the way of modeling and the nature of models are kept fully compliant with business requirements of Credit Suisse. The solution is required to be fully declarative, minimal, decentralized, formal (in a transparent way) and automatable at the same time. From the point of view of theory AGT analysis techniques are specialized for the given class of problems.

The paper is organized as follows: Firstly, we show how laws, regulations and rules can be mapped to the notion of security, risk and compliance and we sketch how these qualities can be measured based on a modeled business process. Secondly, we introduce the notion of a business continuity management system and reflect the corresponding situation at CS. Thirdly, we present a simplified version of a loan granting process as an example of a critical business process at CS and draw the link to an underlying BCMS. Fourthly, we give a short introduction to algebraic graph transformation and reference subobject transformation systems that we are going to use to reconfigure the loan granting process in case of a failure of one of its parts. Fifthly, we demonstrate how a concrete loan granting process model can be analyzed and optimized in an efficient and effective way that fulfills security, risk and compliance issues as a real-world side constraint. Beyond that, we illustrate how case based decisions can be supported. Finally, we draw our conclusions, point to issues of future work and mention some related work.

2 Laws, Regulations, Rules

Laws, regulations and rules determine the degree of freedom and possible boundaries a bank can exploit or needs to respect. They do limit or enforce certain actions and organizational structures.

In the context of this study we like to put our focus on concrete requirements regarding security, risk and compliance derived from laws, regulations and rules. In a first step, we will present today's understanding in the banking environment which is best-practice driven. In a second step, we will point out our understanding which is more aligned towards formal methods. We use this understanding in the following sections to discuss the handling of a loan granting process by the help of an implemented business continuity management system.

2.1 Security

From a best-practice point of view at CS, security can be understood as a set of services. These service encompass the protection of persons, assets, physical property, organizational standards, handling notifications of security incidents, handling policy violations, as well as IT security related issues, etc. From a methodological point of view, we consider security as everything that can be proven based on sound models. We assume that the organizational models and corresponding methods are selected and combined in a way that enables fully automated modelchecking. As an example the separation of duties is presented next.

2.1.1 Separation of duties

The separation of duties is a special security requirement. Its primary objective is the prevention of fraud and error. This is realized by disseminating the tasks and associated privileges for a business process among several persons. It can be illustrated as requirement of two signatures on a contract [2].

Its monitoring and enforcement from a best practice point of view can be realized by the help of an organizational policy that defines that no person should handle more than one type of function and that requires contracts to be signed by two different persons. Such a policy is reviewed, enforced and monitored by a security organization.

Its monitoring and enforcement from a methodological point of view can be realized already when building organizational models. Therefore, it comes along as a side constraint during the modeling process. Because models can be build using algebraic graph transformation, such requirements can be automatically enforced as graph constraint checks on the abstract syntax of formal models [3]. This methodological approach has some advantages like better quality assurance and better scalability than the best practice approach. Cognitive business process models [3] that are aligned with their formal counterparts can easily be used by a workflow engine to monitor security rules which is more efficient than using an additional organizational security structure.

2.2 Risk

Credit Suisse considers different types of risk: market risks, credit risks and operational risks. Here, we only focus on operational risks. An operational risk encompasses inadequate or failed business processes, people or systems caused by certain events that lead to financial losses.

From a best practice point of view, operational risks are managed by organizational solutions like committees and forums, processes and standards, indicators, reports, audits, analysis of loss data, estimation of required risk capital, etc. From a methodological point of view, risks can be much better investigated using simulations of possible failures and their consequences based on sound organizational models. We claim that the case of business processes failures can be backuped to a certain extent by emergency procedures in the context of a business continuity management in particular.

2.2.1 Emergency Procedures

We define emergency procedures (EP) as special micro business processes that are put in place in case that an IT application, a person or a database is not available. It is usually a work-around to guarantee a minimum availability of business services or a certain quality of service.

2.2.2 Business Continuity Management

According to EPs, we define a business continuity management to be a special management function that takes care of emergency planing and handling to ensure that a bank is not going out of business in case of major failures in its critical business processes.

2.3 Compliance

From a best practice point of view at CS, compliance means conforming to a specification or policy, standard or law. A famous example in this context is the Sarbanes-Oxley Act [4] which is about the accuracy of financial statements and the corresponding top management responsibility. It is not always fully defined of how to comply to certain regulations. From a methodological point of view we define compliance as a relationship between certain norms and organizational models and as a relationship between organizational models and the real-world situation. Because a real-world situation does not have to be in line with a model, tests need to be performed to check whether the concrete situation always conforms to the model. As examples, we like to point to the behavior of people inside the bank regarding information barriers and outside the bank regarding agreed payment plans.

2.3.1 Information Barriers

Information barriers exists between different divisions of a bank for various purposes. At CS such divisions are investment banking, asset management, private banking and shared services. The main purpose is to make sure that confidential information is not passed from the private side of the bank to its public side.

For example, a person belonging to the investment banking can act as a broker-agent for a client on behalf of a person belonging to the private banking. According to usual information barriers, such an agent is only allowed to access client data relevant to the transaction but it might be the case that he does access other data too.

2.3.2 Payment Plans

Payment plans in the context of granted loans need to be monitored to see if clients actively comply to the plans. A plan as a business model does not assure that the concrete behavior of a client will conform to it.

3 Business Continuity Management

Business Continuity Management (BCM) is introduced here according to the BS 25999 [5] by taking two different perspectives: the first is a general one, the second a specific one, focussing on the concrete situation at Credit Suisse.

From a general perspective BCM is build on the code of practice, the BS 25999-1:2006 standard, introducing the notion of a Business Continuity Management System (BCMS). The British Standard Institution (BSI) updated this standard using feedback from industry. The BS 25999-2:2007, published November 2007, summarizes the specifications for a BCM. According to Boehmer [6] more than 5000 industrial ideas have been incorporated during this update, this standard is setting out a high level of maturity. Key elements of a BCM are the notion of a disaster, of a business risk, of a critical business process, of a disaster recovery plan, of a business continuity plan, and of the maximum tolerable period of disruption (MTPD). We further define the maximum acceptable outage (MAO), the recovery time to objective (RTO), and the time to recover (TTR).

A disaster is an unforeseen event having a disruptive impact on a critical business process of a company. A business risk is the risk that a disaster potentially has on the business model of a company. A critical business process of a company is a business process the company is running that if interrupted and not recovered in a certain time span causes the company to go out of business in the worst case. A disaster recovery plan of a company is a plan that defines how to recover from the failures in a critical business process that have been caused by a disaster. A business continuity plan is a plan that is applied after the disruption of a critical business process to deliver a minimum level of business activity in order to guarantee the survival of the company during the time the business recovery process is executed.

The MTPD is the maximum time a company can survive without a minimum level of business activity. The MAO is the maximum period of downtime within which the business activities and functions must be fully recovered before the outage compromises business objectives and overall operational viability. The RTO is the maximum time allowed following disaster declaration to return the failed business and IT processes to a minimum level of activity. The TTR is the time taken to fully recover the IT and Business processes.

The BS 25999 is reactive in nature. It comes into play once a catastrophic event has happened. An important control used in the context is the MTPD that defines the tolerable downtime between the disruption of a critical business process and the availability of a minimum level of business activity for this process, defined as RTO, either caused by the business continuity plan or the business recovery plan. It will be assumed that the business continuity plan and the business recovery plan are started simultaneously once the disruptive event has happened. The control $RTO \leq MTPD$ is one measure to evaluate the effectiveness of a BCM because a company will go out of business if it is not able to partially recover from an unforeseen and disruptive event in the time span given by the MTPD. As a second control $TTR \leq MAO$ can be used. It is a measure to evaluate that the IT and business processes are fully recovered in the time span given by the MAO. Any BCMS includes those business processes that are vital to the company.

From the concrete perspective of Credit Suisse, the bank's BCM can be looked at from a strategic and an operational point of view. In the first case, it coordinates according to the bank's global policy business continuity activities including information gathering, planning, implementation, testing as well as crisis management, to assure survivability of the bank in case of a major operational disruption, crisis and disaster. In addition to that IT disaster recovery differentiates itself from an availability management, by the severity of events and non-resolvability with ordinary management techniques and decision making authorities. In the second case, it consists of global and regional concepts and templates defining concrete tasks, a readiness assessment of the implemented BCMS and an established BCM reporting.

In detail, Credit Suisse's BCMS differentiates between a disaster, a crisis, a major incident and an incident. A disaster is an event that is primarily handled by the activation of the business recovery plan. A crisis is an event that requires critical decisions that cannot be resolved with ordinary management techniques. An incident is an event that may lead to a disruption of a critical business process or a low quality of service of a critical business process. A major incident is an aggregation of events that constitutes a group of incidents for which the consequences might be unknown.

The purpose of Credit Suisse's BCMS is to implement and maintain the organization's resilience to disruption, interruption or loss in supplying critical products and services in the event of a major operational disruption. The policy for BCM regulates roles and responsibilities as well as implementation and maintenance of planning, analysis, readiness assessment, communication and training aspects and regulates crisis management.

Credit Suisse's solution today is primarily an organizational solution. The current BCMS is based on a mixed plan and meta-plan-like concept that defines procedures of how to establish concrete plans in the case of a disruption of a critical business process. This plan and meta-plan-like concept is complemented by an organizational structure intended to handle emergency situations, characterized by check-lists and ad-hoc management procedures. The response time in this context is not always known and automatically generated decision alternatives for the given emergency situation are not available. Therefore, the current approach does not scale well. It is underspecified and cannot be smoothly split into orthogonal models. It focuses primarily certain types of disasters, not failures in business and IT processes. Therefore, combinations of failures caused by different disasters are not reflected as such. It lacks decision support and dynamic flexibility depending on the emergency situation. It does not have a sound concept of handling failures at different level of granularity and abstraction. Because of its organizational and informal nature and because continuity fragments are not available as such, no optimization can be performed and case based decisions cannot be supported.

4 A Loan Granting Process

A loan granting process is presented here as a critical business process a bank is running. In a first step, it is introduced from the general point of view of Credit Suisse. In a second step, a more concrete model is introduced that is discussed in the following section 6 more formally. At the end, this model is put in the context of a BCMS.

4.1 Scenario

Credit Suisse (CS) runs its lending business actively. One business objective is to increase the client profitability. To realize this objective the full potential of an existing bank-client-relationship needs to be known to realize possible advantages for both sides. Therefore, a professional advisory service and individual financial solutions tailored to a client's personal situation are offered. Other business objectives are to strengthen the loyalty of existing clients and to acquire new clients.

Potential clients are natural or legal persons. For economic reasons a certain minimum business volume as well as a certain diversity of business activities between the bank and a client is expected.

For security reasons the customer relationship management process, the credit approval and administration process and the credit monitoring process are separated. Therefore, there are four distinct parties in a loan granting process to a client: the client relationship manager, the credit advisor, the credit officer and the credit unit. The client relationship manager is an expert in the concrete client relationship, the credit advisor is an expert in the bank's portfolio of credit products and services, the credit officer is charged with the responsibility of approving credit transactions, and finally the credit unit is handling exceptions in the credit business process as well as the ongoing administration, monitoring and reporting.

A loan is usually granted on a fully secured basis. It can be secured against marketable securities or against banking securities. Depending on the concrete case the percentage of coverage and the date up to which the securities need to be available can change. Because some securities are volatile an ongoing monitoring is needed.

The relationship manager (RM) is responsible for the entire client relationship to assure the highest possible client satisfaction, to develop a long-term business relationship and to optimize the profitability of this relationship. The RM will take care of collateral shortfalls, limit excesses and acount overdrafts.

The credit advisor (CA) is responsible for the lending business products. In particular, the CA is responsible for the profits generated by the lending products. His duty is to promote the lending business, to administrate specific requirements of clients, to assess the credit worthiness and to handle loan applications. The CA will – in cooperation with the RM – provide advice to the client. For example, he will point out the opportunities available regarding the various lending products and services. The CA advices the RM regarding specific conditions of products and services as well as their appropriate handling. He is responsible for the renewal of credit approvals.

The credit officer (CO) is responsible for credit approvals as well as limit accesses or account overdrafts that are beyond the defined tolerances.

The credit unit (CU) is responsible for the ongoing monitoring, reporting and control of running credit processes, their corresponding shortfalls and customer positions. The CU is assuring compliance with credit limits and repayment schedules as well as credit settlements at the maturity date. At CS, the credit monitoring is mainly based on reports automatically delivered by the IT systems in place. The CU can take actions in the event of emerging difficulties during a credit process.

From the point of view of this paper this loan granting process is simplified to match the scope of this study. It encompasses a client, a relationship manager, a credit advisor and a credit officer. The process is characterized by steps that are performed manually, steps that are executed automatically and steps that are hybrid. The view on the process is the one taken from a workflow engine that runs workflow instances based on their workflow scheme. The notation used is the one for event driven process chains, but in a slightly modified version to fit the requirements of the presented scenario in a better way. The process itself covers the whole lifespan of a granted loan, starting with the demand for a loan, ending with its finished payment plan.

We assume that, once a client (C) arrives at the bank, he will be asked a couple of things by the RM (functions F1-F4 in the workflow model in Fig. 2). Firstly, he needs to identify himself and the RM will try to make a first estimation about the possible customer value in an assumed business relationship. Secondly, the RM will record the client's demand. All data is entered into IT systems by the RM. In the following the credit worthiness and a customer rating for C is calculated automatically by two different applications (F5-F6). Based on the rating the CA will make a decision if C will be accepted for a loan (F7). In the next step, an optimized product is created by the CA and the RM for C (F8). Afterwards, the RM creates a contract for C (F9). This contract has to be signed by C and the RM. The credit officer (CO) needs to approve the contract (F10-F12 in Fig. 2). Here the 4-eye principle applies for security reasons because the RM and the CO are not allowed to be the same person. Afterwards, the bank pays the granted loan to C, and C is paying the credit back according to a payment plan up to the moment the contract will be closed (F13-F15).

From the point of view of a BCM this loan granting process can be discussed looking at certain failures that can happen in the process. Because we like to introduce fully automated continuity techniques respecting security, risk and compliance issues as real-world side constraints the perspective on the process is the one of a workflow engine that implements these techniques. Further, we like to base our discussion on a running process instance, not only on the underlying process scheme as it is done today in the context of BCM. This leads to highly optimized reactions towards certain failures in a critical business process. We assume that for most steps in a critical business process continuity fragments are available to backup those steps. We further assume that the elements that can fail in a process are people, applications and databases. Depending on concrete failures, a workflow engine can select an emergency process based on the calculated set of all continuity processes. It can reconfigure the running process instance in order to optimize time and cost functions while fulfilling security, risk and compliance side constraints.

The critical business process about granting a loan is introduced next from the point of view of a workflow engine using a slightly modified version of an event driven process chain (EPC) [7].

This type of EPC consists of business functions, events, organizational entities and applications executing business functions, as well as the dataflow into and out of a business function from or to certain data storage units. Such storage units are either databases, or, alternatively, organizational entities, like concrete persons. Because we take the perspective of a workflow engine, this scheme will run as an instance when performing a concrete business process. As such, a workflow instance owns local storage to remember certain data values. Data values that are kept in local storage of the workflow instance are modeled by dark-blue boxes. Data values that are not kept in local storage are put into light-blue boxes. This allows to cover automated and non-automated parts of a business process by the help of one single process model. In case that a data value is still known to the workflow instance, it has not to be re-loaded from a database or entered by a person. In case that a failure occurs, data values can be re-loaded on demand. We like to name this a data-flow oriented EPC from the point of view of a workflow engine, or in short: WDEPC. A central advantage out of this understanding is that business functions can be shifted back and forth in the process chain depending on the data values they require. So, in contrast to today's understanding of EPCs, data flow dependencies can be handfeed orthogonal to event chain dependencies.

Further, the notion of local storage of a workflow instance enables work-arounds of temporarily non-available data storage units. A business function is performed by a person, an application or any combination out of these. Data values not relevant to the workflow are abstracted.

4.2 Business Process Model

The presented business process has to meet certain objectives. There are primary and secondary goals as shown in Fig. 1. From the point of view of CS, this process needs to generate a contract and leads to a long-term client-bank-relationship. Relevant attributes regarding the contract are the time required and the realized costs, regarding the relationship the relevant attribute is the measurable client loyalty. It further requires to respect certain security, risk and compliance requirements.

In the given case, security will be proven by the help of a graph constraint check assuring the four eye principle when signing the loan contract. Risk will be simulated from a perspective of the workflow scheme discussing possible failures and recovery strategies in the limits of a 48h MTPD BCM baseline ($RTO \leq MTPD$). Finally, compliance is assured by testing the workflow instance against the real-world situation.

Figure 2 shows the WDEPC language artifact for the presented loan granting process: WDEPC LG. The diagrams are divided into five columns. Starting on the left, there are data storage units. The next column contains the corresponding data items, where solid lines indicate that the data is also locally cached within the workflow engine. This allows us to cover automated and non-automated parts of a business process by one single declarative process model. The third column consists of the business functions, the fourth contains the events and finally the fifth column consists of the organizational entities, i.e. persons or software applications.

While this process model shows a fixed order of certain functions many of them are not directly dependent on each other. But they are partly dependent on the data that is produced by previous

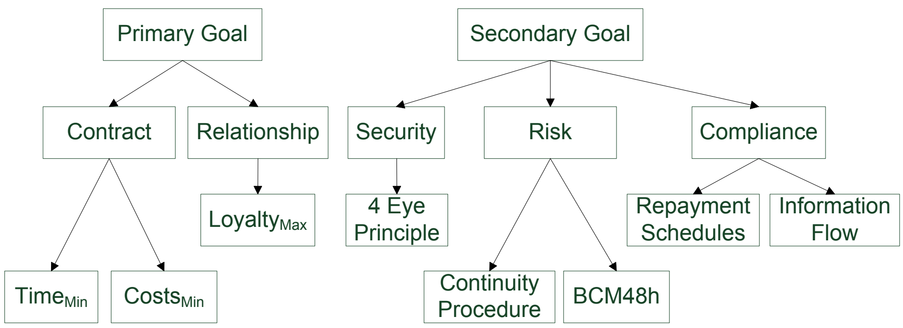

Figure 1: Primary and Secondary Objectives

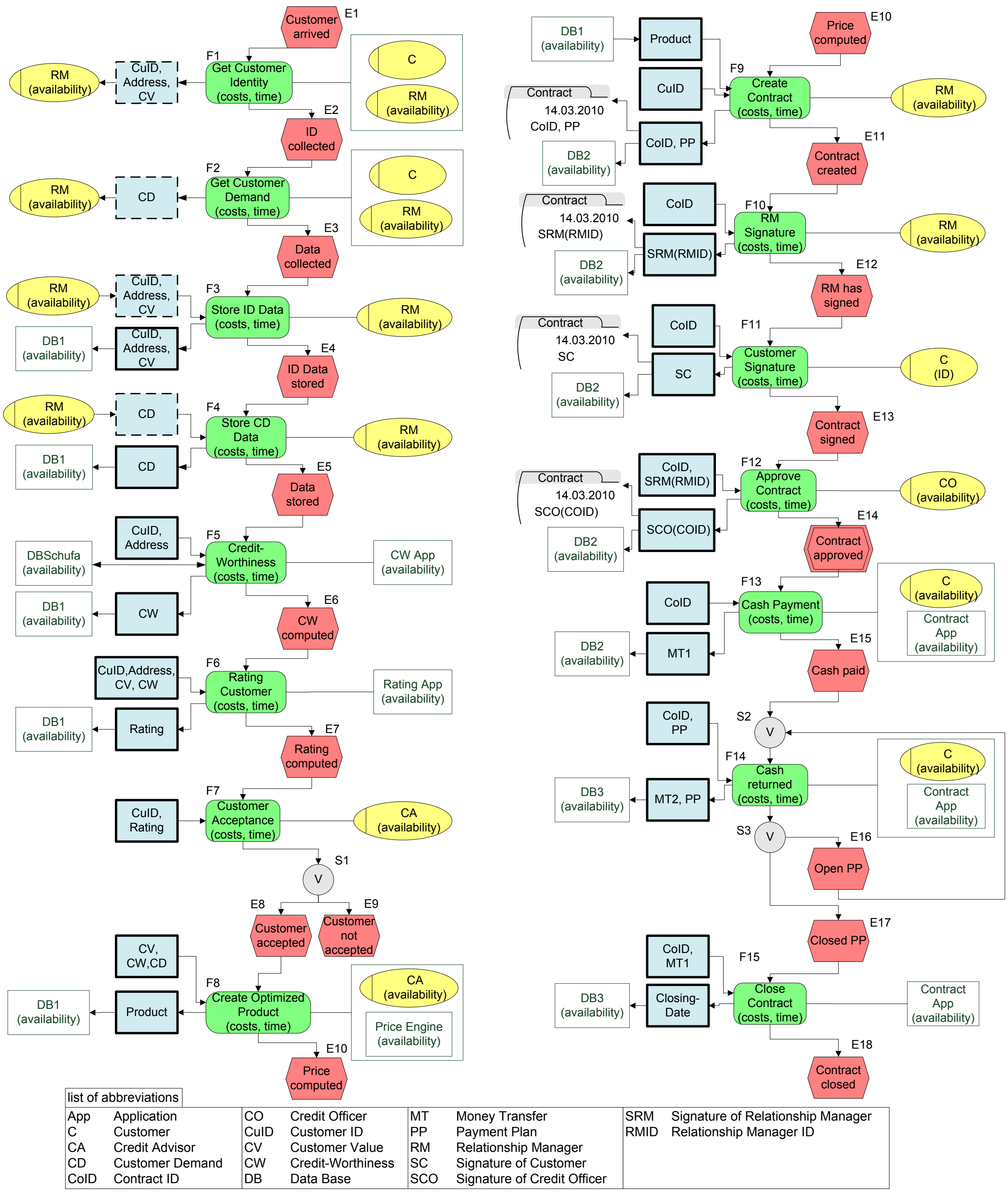

Figure 2: Workflow *LG*

steps. Look e.g. at functions "F1" to "F4" for getting and storing certain customer data. Clearly, "F1" needs to be executed before "F3" and similarly "F2" before "F4", but there are no further dependencies. Thus, leaving out the synthetical events in between we are interested in all permutations of such steps fulfilling the data dependencies to generate valid process variants. However, there are some specific events like "E8", "E9" or "E14" which restrict possible permutations in the sense that they play the role of a boundary functions cannot pass when shifted up or down. Critical parts of the workflow are e.g. at functions "F10" where the contract is signed by the relationship manager and "F12" where the contract is approved by the credit officer. Here, the four-eye principle applies as a security requirement and demands that these functions have to be performed by two different persons. For this reason, also the continuity processes have to ensure this requirement. It can be codified as a graph constraint.

5 Algebraic Graph Transformation

In order to support the development of business recovery and business maintenance plans for different scenarios we propose to apply algebraic graph transformation [8], which is a formal, visual and intuitive technique for the rule based reconfiguration of object structures. Graph transformation offers analysis techniques for dependencies between transformation steps, which specify the actions of a business process in our scenario. This allows us to show how possible modifications of the process steps can be automatically computed.

From the formal point of view a graph grammar $G = (TG, R, SG)$ consists of a type graph TG, a set of transformation rules R and a start graph SG. The type graph specifies the structure of possible object structures and the rules constructively define how models are modified. The start graph is typed over TG and is the starting point for each transformation. Each graph $G = (V, E, src, tgt)$ is given by a set of vertices V, a set of edges E and functions $src, tgt : E \to V$ defining source and target nodes for each edge. Graphs can be related by graph morphisms $m : G_1 \to G_2$, where $m = (m_V, m_E)$ consists of a mapping m_V for vertices and a mapping m_E for edges, which have to be compatible with the source and target functions of G_1 and G_2. Note that we can also use modeling features like attribution and node type inheritance as presented in [8].

$$\begin{tikzcd}
L \arrow[d, "m"'] \arrow[dr, phantom, "(PO_1)"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d] \arrow[dr, phantom, "(PO_2)"] & R \arrow[d, "m^*"] \\
G & D \arrow[l] \arrow[r] & H
\end{tikzcd}$$

The core of a graph transformation rule consists of a left-hand side L, an interface K, a right-hand side R, and two injective graph morphisms $K \xrightarrow{l} L$ and $K \xrightarrow{r} R$. Interface K contains the graph objects which are not changed by the rule and hence occur both in L and in R. Applying rule p to a graph G means to find a match m of L in G and to replace this matched part $m(L)$ by the corresponding right-hand side R of the rule. By $G \overset{p,m}{\Longrightarrow} H$ we denote the graph transformation step where rule p is applied to G with match m leading to the result H. The formal construction of a transformation step is a double-pushout (DPO), which is shown in the diagram above with pushouts $(PO1)$ and $(PO2)$ in the category of graphs. D is the intermediate graph after removing $m(L)$ and H is constructed as gluing of D and R along K.

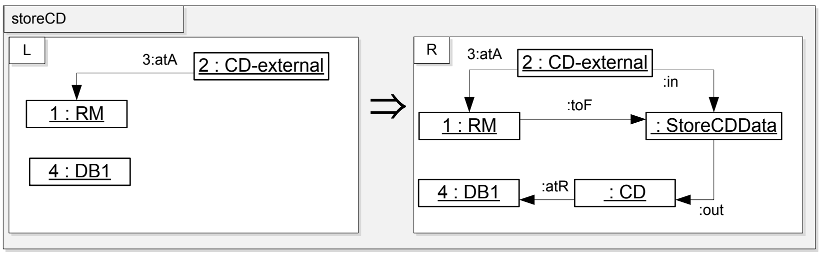

Figure 3: Rule *storeCD*

Fig. 3 shows the rule *storeCD*, which specifies the function “Store CD” of the EPC for the loan granting process in Sec. 6. Since the rule *storeCD* is nondeleting we have that in this case $K = L$. This will also be the case for all derived rules in our scenario. The effect of the rule is the creation of a node with type “StoreCDData”, which corresponds to the process function “Store CD Data”, where CD abbreviates “Customer Demand”. Furthermore, edges are inserted to connect the new function node with its actors and data elements. The morphisms $l : K \to L$ and $r : K \to R$ of the rule are denoted by numbers for nodes and edges, i.e. each number specifies a mapping between two elements.

6 Analysis and Optimization

At Credit Suisse continuity plans are defined only for certain fixed scenarios at a macro level, even so there are lots of small failures in such a scenario. That leads to problems when more than one continuity scenario is going to come up at the same time, or when failures that are assigned to different scenarios mix up in a new scenario. In addition to that Credit Suisse' continuity plans are to a large extent not operational but need to be instantiated towards a concrete situation. That makes it impossible to check their effectiveness and efficiency. Because continuity processes based on a given continuity plan are created ad-hoc at Credit Suisse their effectiveness and efficiency cannot be evaluated either. So, Credit Suisse knows very little about the quality of its own continuity procedures. That drives insurance costs. It is like you carry wood with you for life-boats as well as a construction manual but you never checked how many boats you will get at the end and in which order people can be evacuated best.

By looking at the human-centric business process model of the presented loan granting process we like to show how to address these problems in a methodological way. In contrast to Credit Suisse' approach which is top-down we like to go bottom-up by defining continuity snippets for the presented business process. A second difference is that Credit Suisse' approach is static, which means it only addresses one very specific scenario, whereas ours is flexible and can address a wide range of different scenarios by applying continuity snippets to emerging failures in unforeseen ways by the people in the field. A third difference is that Credit Suisse' approach does not scale well, it always requires to execute the whole continuity plan, whereas our approach can dynamically focus very specific parts of a scenario like minimal invasive medical operations. It only needs to apply continuity snippets that are linked to concrete failures, not to whole scenarios. Therefore, we use the snippets as well as security rules like the one for the four-eye principle to generate the universe of business continuity processes given the universe of all functional valid process variants. Once this universe is available single continuity processes as well as the whole set can be checked regarding their effectiveness and efficiency. We will focus on the generation of valid process variants as well as possible continuity processes. The check for their effectiveness and efficiency will be left for future work.

6.1 Graph Grammar for a WDEPC

Business process models given by EPCs often consist of chains of functions. The intermediate events imply virtual dependencies of consecutive functions, even if these dependencies do not exist in the real process. In the following, we describe the construction of a graph grammar GG to define the operational semantics of the LGP. Thereafter, we show how dependencies that are mainly not caused by the events but by the dependencies on data elements and actors are computed using the constructed grammar. The reconstruction of the graph grammar can be performed automatically based on the underlying abstract syntax of WDEPC-models that can be created during the modeling phase by e.g. using a visual editor that is generated by the Tiger environment [9, 10]. Furthermore, we can define the functional requirements as an additional graph rule, by stating which data elements have to be created in the process for a certain combination of events. In the presented example we have the requirement that either the customer is not accepted or the contract shall be approved and a closing date has to be set.

Given a workflow model in form of a WDEPC the corresponding graph grammar $GG = (TG, RULES, SG)$ is reconstructed as follows:

- The type graph TG contains the nodes and edges of the WDEPC, where nodes with the same label that occur several times in the WDEPC are identified and occur only once in TG. Analogously, edges with the same source and target node are identified and appear only once in TG.

- The start graph SG consists of the nodes for the actors, i.e. the organizational entities, and the resources only.

- Each function is translated into a graph rule (see e.g. function "Store CD Data" Fig. 2 and its corresponding rule in Fig. 3). The left hand side of the rule contains the actors, the input data elements with its resources and the edges between these elements. The right hand side additionally contains the function node, the output data elements, and the edges that connect the nodes as given in the WDEPC. A rule may be extended, if connected to a special event as specified in the next item.

- An event is a control event, if its frame has an additional line or the event is the successor of a fork node. If a control event is the predecessor of a function, then the RHS of the rule for the function is extended by this event. If a control event is a direct or indirect predecessor of a function then the LHS and RHS of the rule are extended by this event. An function is an indirect predecessor of an event, if there is a path of sucessor edges from the event to the function.
- Finally, there is a special rule "neededData", which creates all data elements that are not created by any other rule. Thus, the RHS contains all data elements and their links to resources, which are in the set N specified below. The LHS is given by the resources that are connected to the links in the RHS and these resources are also in the RHS.

  $N = TG \setminus (\bigcup_{r_i \in RULES} R_i)$, where each graph is flattened to a single set containing its nodes and edges.

Figure 5 shows the type graph TG for the graph grammar GG_{LG} as reconstructed from Fig. 2. Some nodes and edges are depicted several times in order to improve the layout, but they are identified as described before. Those nodes and edges, which appear again are marked by a grey fill resp. line color as e.g. the node "RM" or the edge "atA" from "CuID" to "RM" in the top left area of the figure. This type graph directly corresponds to the abstract syntax of the WDEPC. This type graph is itself typed over the meta type graph TG_{Meta} for WDEPCs, which is shown in 4. The reason why we do not directly use the meta type graph is that the we apply the theory of subobject transformation systems, where an injective typing is required. This means that each rules LHS and RHS have to be included in the type graph, i.e. the type graph is the super object for each considered graph. Since the additional continuity snippets in Sec. 6.3 introduce some additional nodes and edges the type graph is extended accordingly.

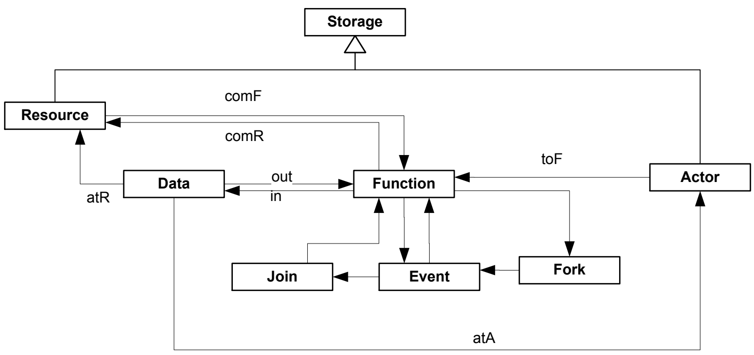

Figure 4: Meta Type graph for graph grammars for WDEPCs

The adjacent edges of the event nodes are not used by any rule in general, because the do not provide additional information required for the analysis. The reason is that the relevant events are contained in the same rule as the function, which is reading or creating the event. A WDEPC can be simulated by applying the generated rules to the start graph according to the order in the WDEPC. Each intermediate graph represents the current state for the execution of the process and each rule application ensures that not only the necessary input data elements are available but also that they are visible via the involved resources to which the particular actor has access. If the RHS of the special rule "neededData" is not empty, we have that the given WDEPC contains at least one function that cannot be executed, because the needed Data was not produced by any function. This means that the WDEPC is not sound in the way that all parts may be executable in at least system run.

The dependencies between the functions of an WDEPC can be analyzed by the dependencies between the rules itself, because the derived graph grammar fulfills the additional conditions of a subobject transformation system - a graph grammar, where each rule component is injectively

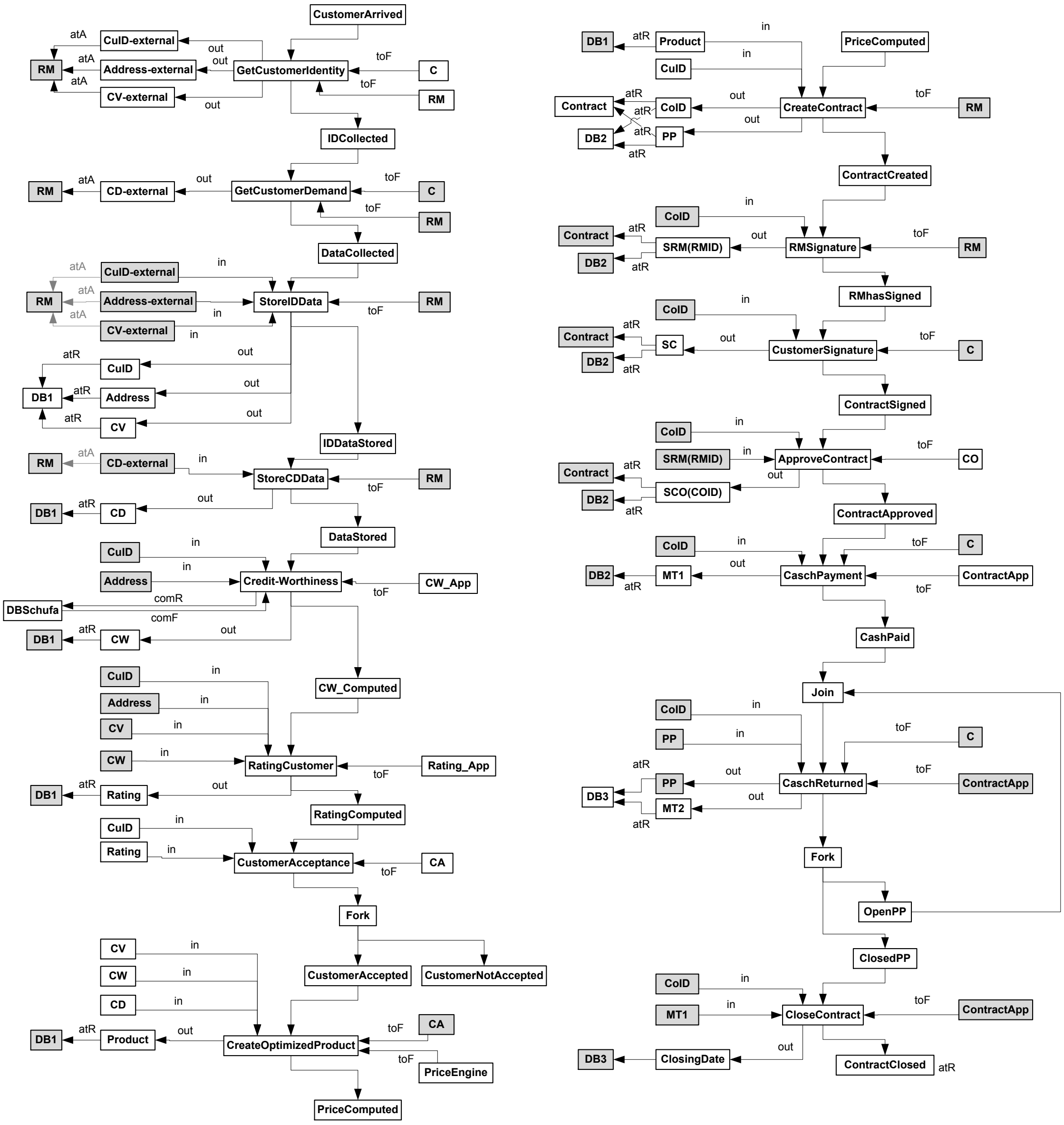

Figure 5: Type graph of the graph grammar GG_{LG}

typed. For this reason, we can apply the efficient techniques especially developed for the analysis of dependencies in processes [11, 12, 13].

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

The following figures show the rules of the graph grammar GG_{LG} for the critical business process modeled by the WDEPC LG as shown before. Each rule corresponds to an equally named function in LG.

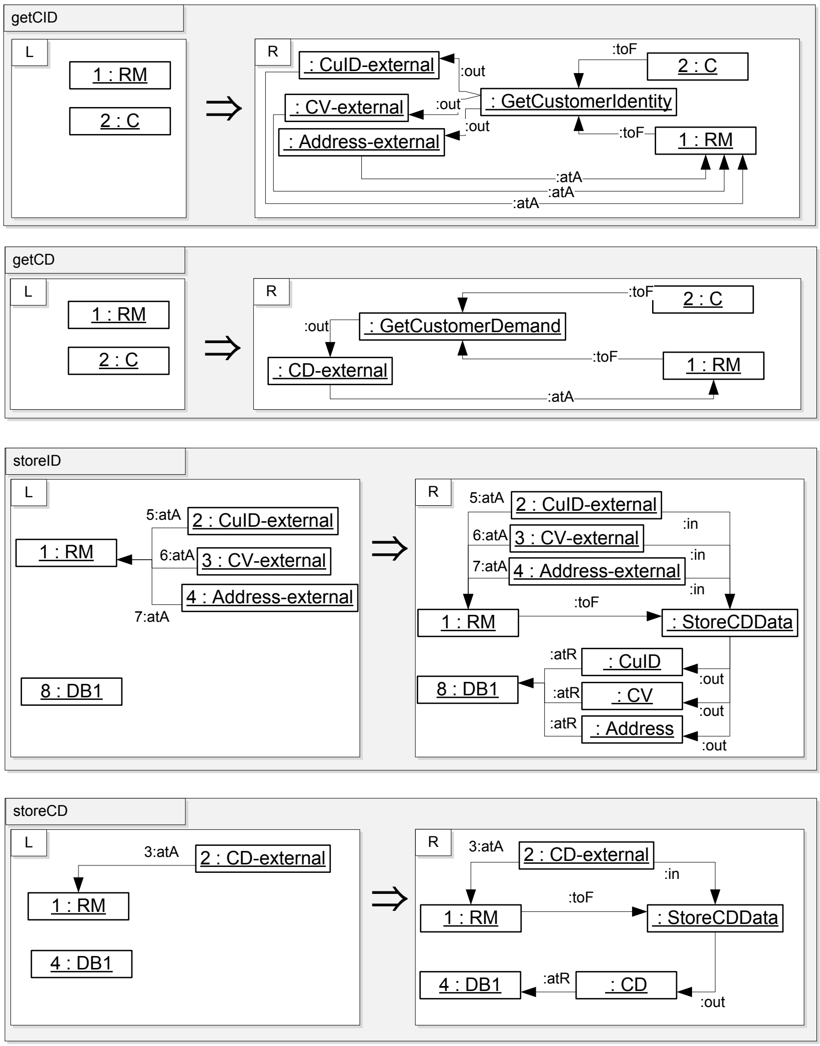

Figure 6: Rules of graph grammar GG_{LG} - Part 1

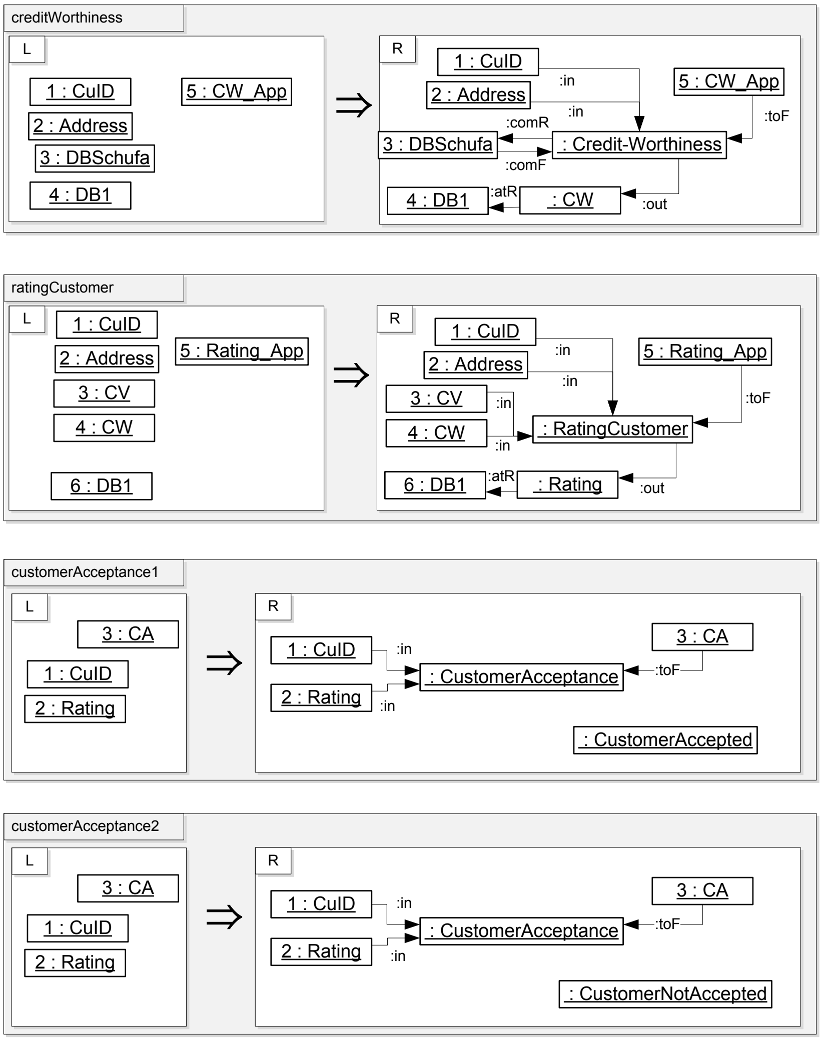

Figure 7: Rules of graph grammar GG_{LG} - Part 2

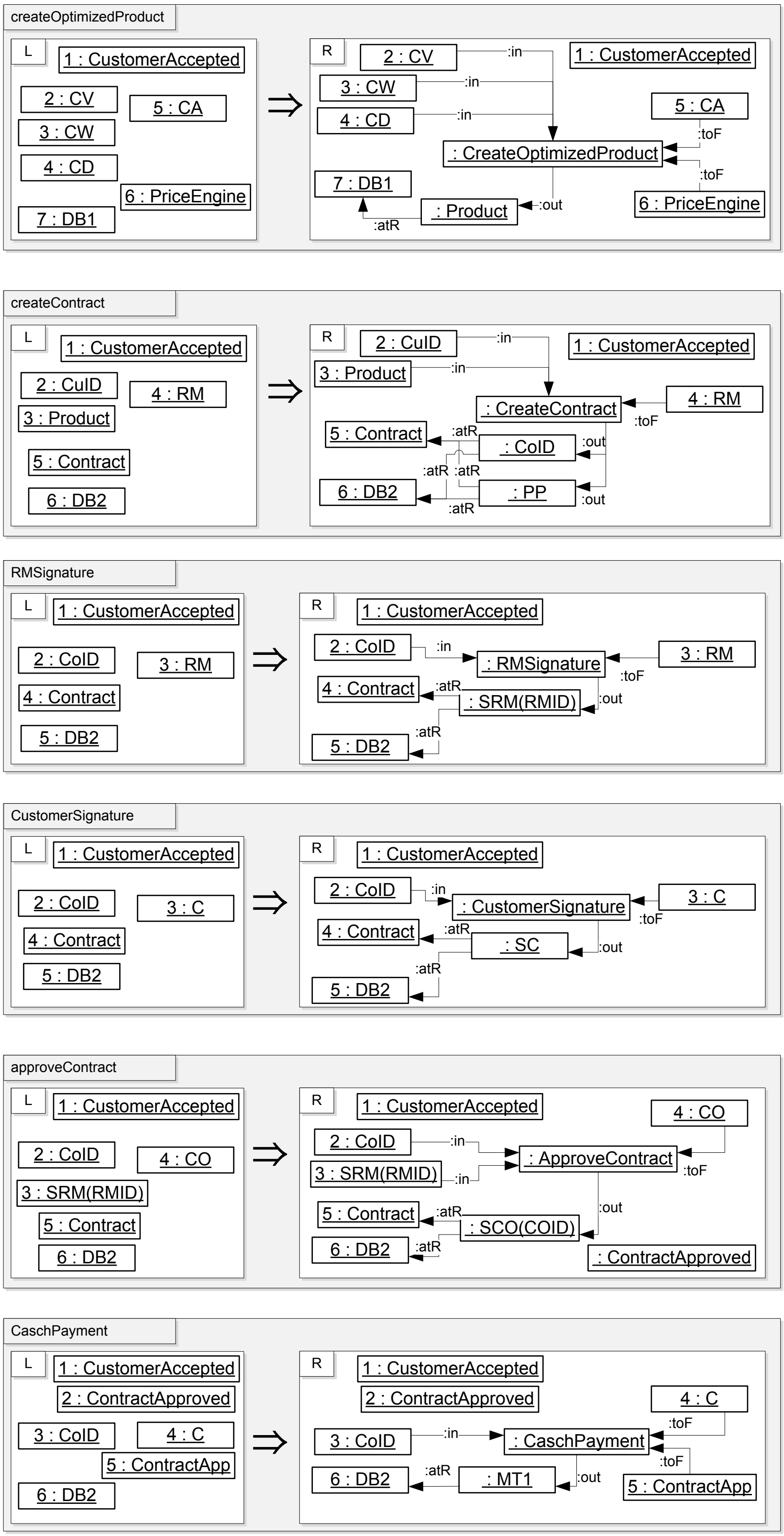

Figure 8: Rules of graph grammar GG_{LG} - Part 3

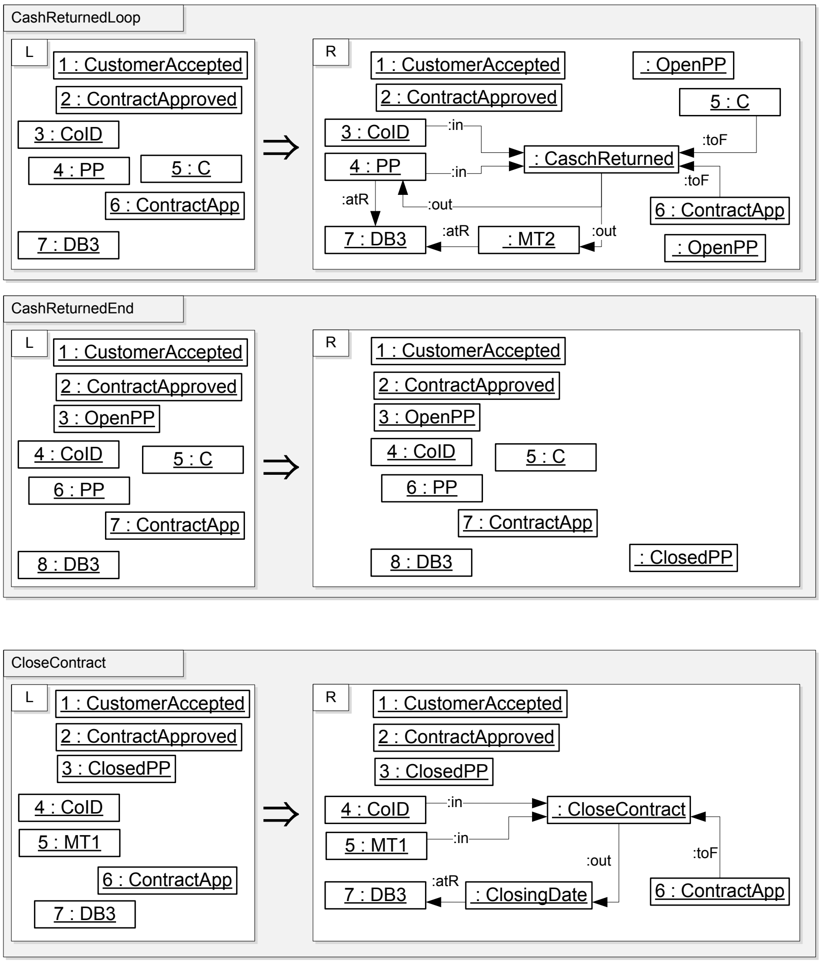

Figure 9: Rules of graph grammar GG_{LG} - Part 4

6.2 Computation of Dependencies

The first step for the generation of business continuity processes is the analysis of existing dependencies between the single steps of a process. Here, we focus on data-flow dependencies between functions. We therefore remove the dependencies caused by the syntactic events between the functions and only keep specific event-dependencies, which are marked as e.g. "E14" in Fig. 2 or which are used for a control structure as e.g. "E8" and "E9". This analysis allows for possible process paths that show a different order of the steps and possibly enable the exchange of certain parts with continuity fragments that were not possible in the standard order. Furthermore, the process is kept flexible which improves the usability for the actors assuming that the execution of the process is supported by a workflow engine.

Consider the first four functions "Get Customer ID", "Get Customer Demand", "Store ID Data" and "Store CD Data". The corresponding rule of "Store CD Data" is shown in Fig. 3 and the only dependencies for the corresponding rules $p_1 = getID$, $p_2 = getD$, $p_3 = storeID$ and $p_4 = storeCD$ in GG_{LG} are: $p_1 <_{rc} p_3$ and $p_2 <_{rc} p_4$, where "rc" denotes read causality. This means that p_3 uses a structure that is created by p_1 and p_4 uses structures that are created by p_2. Now, the WDEPC LG requires a sequential execution. However, the dependencies based on the rules also allow that first the demand of a customer is determined and stored and thereafter, the necessary identification information is collected and stored. This means that the four steps can be executed in several ways - all together 6 variants - only the partial order given by the dependency relation $<_{rc}$ has to be respected. The relation manager shall be able to act upon the customer preferences and upon the course of conversation, such that any of the possible interleavings should be possible. Of course, the possible interleavings can also be achieved by modifying the EPC, but during the modeling of an EPC for a business process several possibilities of concurrency will not be detected, because the real actors are asked to specify the standard execution. Fortunately, the computation on the basis of the corresponding rules can be performed automatically, because it uses the dependency relations of the underlying STS and these relations are computed statically, i.e. without any need for executing the steps.

Now, have a look at the end of the example process LG where functions "Customer Signature" and "Approve Contract" occur. The corresponding rules are $p_{11} = customerS$ and $p_{12} = approve$. There is no dependency between these rules implying that the customer may sign the contract before or after the contract is approved by the credit officer. Consider the case that the customer may want to see both signatures on the contract before he signs. Thus, this inverse order is relevant. Note that it is not trivial to find this partial independence while building an EPC model by hand.

6.3 Computation of Alternatives

In order to construct complete continuity processes for a combination of failures we first show how process fragments are replaced and composed:

Composition of Process Parts Consider that we have process parts $P1$ and $P2$. They are composable, if first of all the start event of $P2$ occurs in $P1$ - this is the gluing point of the composition and we denote the new part by $Q = P1; P2$. Furthermore - in order to have that $P1; P2$ can be executed - each left hand side of a rule p_x of the corresponding grammar GG_Q has to be included in the start graph joined with the right hand sides of the rules that correspond to preceding steps. This condition is sufficient, because the constructed rules are non-deleting. If $P1$ is already executable then the check can be reduced to the rules of GG_{P2}.

As soon as a resource or an actor is not available the process execution has to be replaced by an alternative execution sequence, which contains suitable alternative process parts, such that the alternative execution is possible and fulfills all requirements. Consider the following failure in the present scenario: the rating application in the WDEPC "LG" is not available, which implies that the function "Rating Customer" cannot be executed. In this case the alternative function "Rating Customer (C)" in Fig. 10 can be executed, where "(C)" denotes that it is a continuity function for a certain failure of a resource. Exchanging function "Rating Customer" with function "Rating Customer (C)" may cause conflicts with other functions. The underlying dependencies with respect to the other functions of the current chain of process steps can be analyzed using the corresponding graph transformation rules. This analysis can be performed statically, i.e. before a failure occurs, and the results can be stored and remain valid during a process execution.

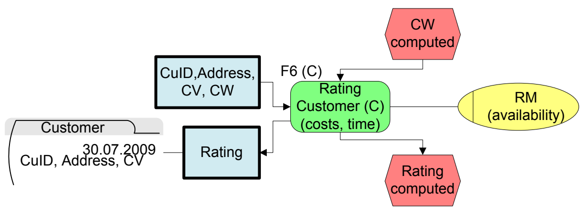

Figure 10: Alternative Function "Rating Customer (E)"

"Rating Customer (C)" needs the availability of "CuID, Adress, CV" and "CW", which are provided by the functions "Get Customer Identity" and "Credit Worthiness". These dependencies are present for the corresponding rules $p_1 = getCID, p_5 = creditWorthiness, p_6 = ratingCustomerC$ as well. We have the following pairs of the relation "read causality": $p_1 <_{rc} p6$) and $p_5 <_{rc} p6$), i.e. $p6$ needs some elements that are produced by p_1 and p_5.

Furthermore, we have to ensure that all elements that are necessary for the succeeding steps of "Rating Customer (C)" are present. Thus, we have to ensure that each element, that is created by function "Rating Customer" is:

1. created by "Rating Customer (C)" as well or
2. not needed by a succeeding step.

The complete business continuity process is constructed stepwise and for each step the following condition (1) ensures that the succeeding steps can access the elements they need. In more detail, the rule $p_i = (L_i \leftarrow K_i \rightarrow R_i)$ of condition (1) below corresponds to the $i(th)$ function of an WDEPC and p_i shall be replaced by the alternative rule $p'_i = (L'_i \leftarrow K'_i \rightarrow R'_i)$. The elements in the set $(R_i \setminus K_i)$ are the nodes and edges that are created by the rule p_i.

$$\left[(R_i \setminus K_i) \cap \bigcup_{j>i} L_j\right] \subseteq R'_i \setminus K'_i \tag{1}$$

Fortunately, this condition is fulfilled for "Rating Customer (C)" in Fig. 10 and we can use this fragment. Furthermore, independent succeeding steps can be moved to preceed the critical function, which delays the execution of the continuity fragment - e.g. in Fig. 11 the steps $a7$, $a8$ are moved in front of $a6$, which is going to be replaced by $a6'$. If the missing resource is available again and the delayed function is still not executed then the original function can be executed instead. This is an important advantage of the automatic analysis capabilities and the generation of possible continuity processes.

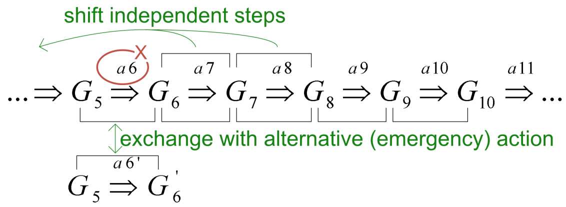

Figure 11: Automatic generation of alternatives

Alternative process parts may contain several steps that furthermore may only replace parts of the original steps or cause conflicts with other steps, which implies that additional alternatives have to be used to build up a complete alternative. In Fig. 12, two alternative parts are composable with the original process by exchanging it with steps $a1$ to $a4$. The step $a2$ is not completely covered by one alternative fragment but by the composition of the two fragments. In order to find optimal continuity processes annotated costs and time values of the functions can be used.

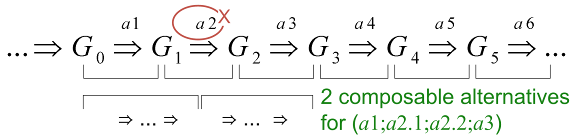

Figure 12: Complex alternatives

We now present the further emergency fragments for the process LG, such that alternative continuity processes can be generated automatically for combinations of failures. For some business functions there are two emergency fragments, depending on the combination of failures that may occur.

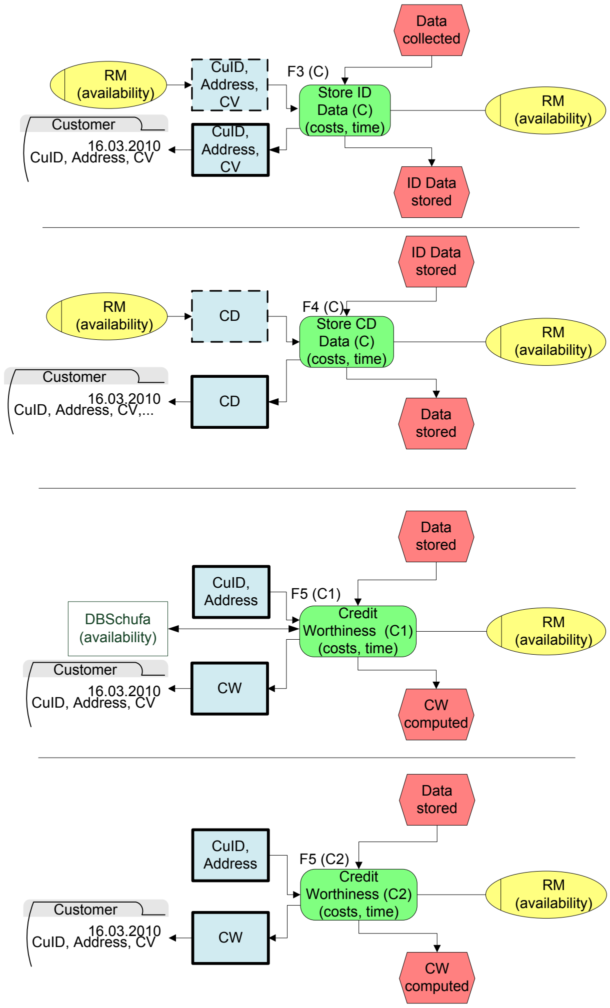

Figure 13: Emergency Fragments - Part 1

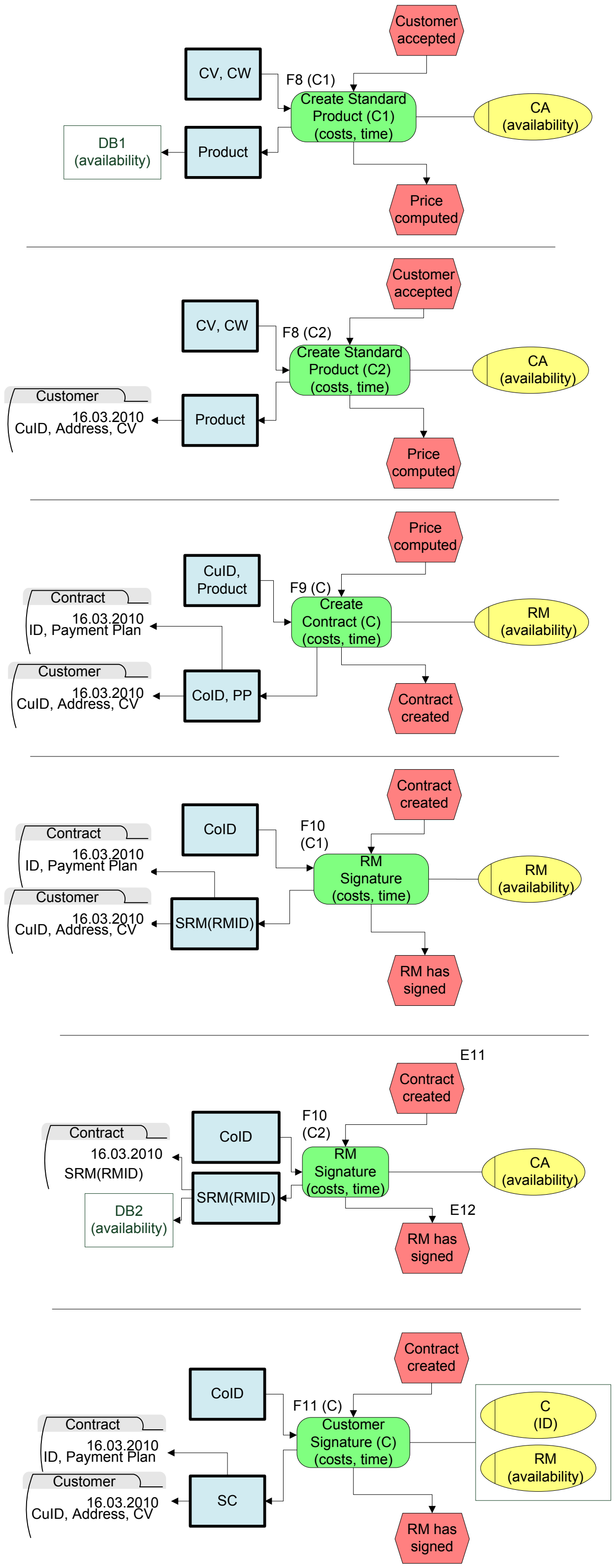

Figure 14: Emergency Fragments - Part 2

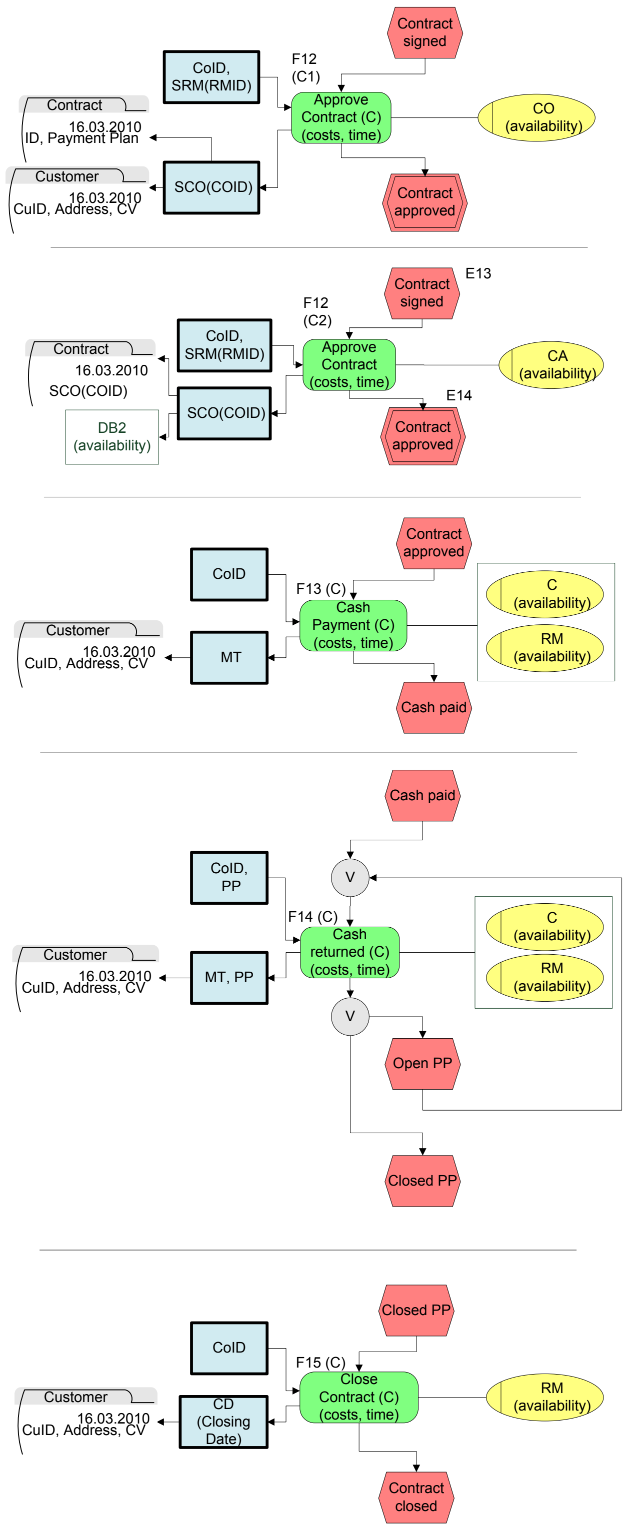

Figure 15: Emergency Fragments - Part 3

6.4 Validation of Objectives

Usually, several security constraints have to be respected by all possible process variants. As an important example we illustrate the handling of the four-eye principle specified by the constraint in Fig. 16. The functions “RM Signature (F10)” and “Approve Contract (F12)” have to be performed by different persons in order to ensure a separation of concerns. The graph constraint is given by a negation of the formal constraint "samePerson", which states the following: If the premise P is fulfilled, then also the conclusion graph C has to be found, i.e. in this case the two functions are executed by the same person as specified by the connecting edges.

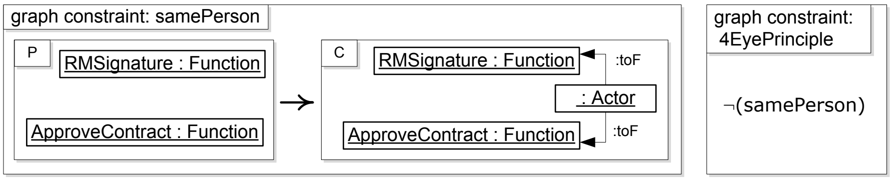

Figure 16: Graph Constraint: 4 Eye Principle

The shown constraint is a meta constraint, i.e. it is not fully typed over the type graph TG of the reconstructed graph grammar. The conclusion graph C contains the a node of the meta type "Actor", for which the type within TG is not specified. Thus the effective constraint is obtained by instantiating the meta constraint into a set of graph constraints typed over TG. This construction is introduced by the following definition using the notion of instantiated graphs, which corresponds to Def. 15 in [12].

Definition 1 (Instantiated Graph)**.** *Given a typed graph (G, t_G) typed over TG_{Meta}, a type graph (TG, t_{TG}) typed over TG_{Meta}, and a partial mapping from G to TG via the span of injective typed graph morphisms $G \xleftarrow{d} D \xrightarrow{type_D} TG$ typed over TG_{Meta}. An instantiation of G within TG is given by $(G, type_G : G \to TG)$, where $type_G$ is an injective graph morphism typed over TG_{Meta} and $type_G \circ d = type_D$ as shown below. The set of all instantiated graphs for G and TG via d is denoted by $Inst(d, G, TG)$.*

$$\begin{tikzcd}
 & TG_{Meta} & \\
 & TG \arrow[u, "t_{TG}"] \arrow[r, phantom, "(=)"] & {} \\
D \arrow[ur, "type_D"] \arrow[rr, "d"'] \arrow[r, phantom, "(=)"] & {} & G \arrow[ul, "type_G"'] \arrow[uul, bend right, "t_G"']
\end{tikzcd}$$

Based on the definition for instantiated graphs we extend this notion the the case of constraints.

Definition 2 (Meta Graph Constraint)**.** *Given a graph constraint $PC(a) = P \xrightarrow{a} C$ with injective a typed over TG_{Meta}, a type graph TG typed over TG_{Meta}, and a partial mapping from P to TG via a span of injective typed graph morphisms $P \xleftarrow{d_1} D \xrightarrow{d_2} TG$ typed over TG_{Meta}. The graph constraint $(PC(a), d)$ is called a meta graph constraint. An instantiated graph constraint of $PC(a)$ is given by $(P, type_P) \xrightarrow{a} (C, type_{C,i})_{i \in I}$, where $(P, type_P)$ is an instanctiated graph in $Inst(d, P, TG)$ and $(C, type_{C,i})_(i \in I)$ is an ordered list of the set $Inst(a, C, TG)$ using the partial mapping from C to TG via the span of injective typed graph morphisms $C \xleftarrow{a} P \xrightarrow{type_P} TG$. The set of instantiated graph constraints is denoted by $Inst(d, PC(a), TG)$.*

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

Note that the set of instantiated graph constraints may contain several constraints but it may also be empty meaning that it is always fulfilled. For the given graph constraint in Fig. 16 we derive exactly one instantiated graph constraint with one conclusion as shown in Fig. 17.

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

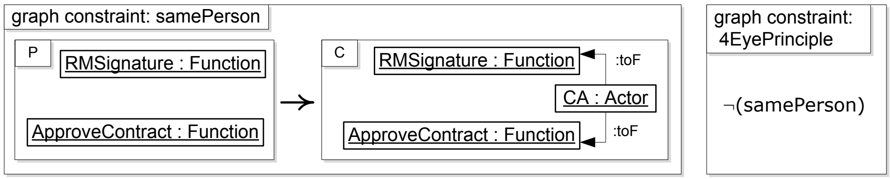

Figure 17: Instantiated Graph Constraint for the Security Requirement: “the 4-eye principle”

If a condition shall be ensured only locally, i.e. for a single function like “Approve Contract”, the constraint can be formulated directly as an application condition for the corresponding rule of the function [8, 12].

In the next step a dependency net is generated out of this grammar as presented in [13], which is given by a place/transition Petri net that purely specifies the dependencies between the steps. This net is used to generate the universe of the execution paths of the variants of the given business process and its corresponding continuity processes.

7 Generated Universe of Continuity Processes

Once the process model given by a WDEPC is analyzed as described above, the reachability graph of the derived dependency net can be generated, where functional as well as security requirements are respected in the way that steps that violate the security requirements are not performed and paths that finally do not fulfill the functional requirements are filtered out. This way, the universe of valid process paths is generated as shown in Fig. 18. Because graph techniques are well suited to check for local requirements we propose to use a machine-centric business process model based on process algebra to check for global requirements like data-flow and information-flow aspects later on as presented in Sec. 8.

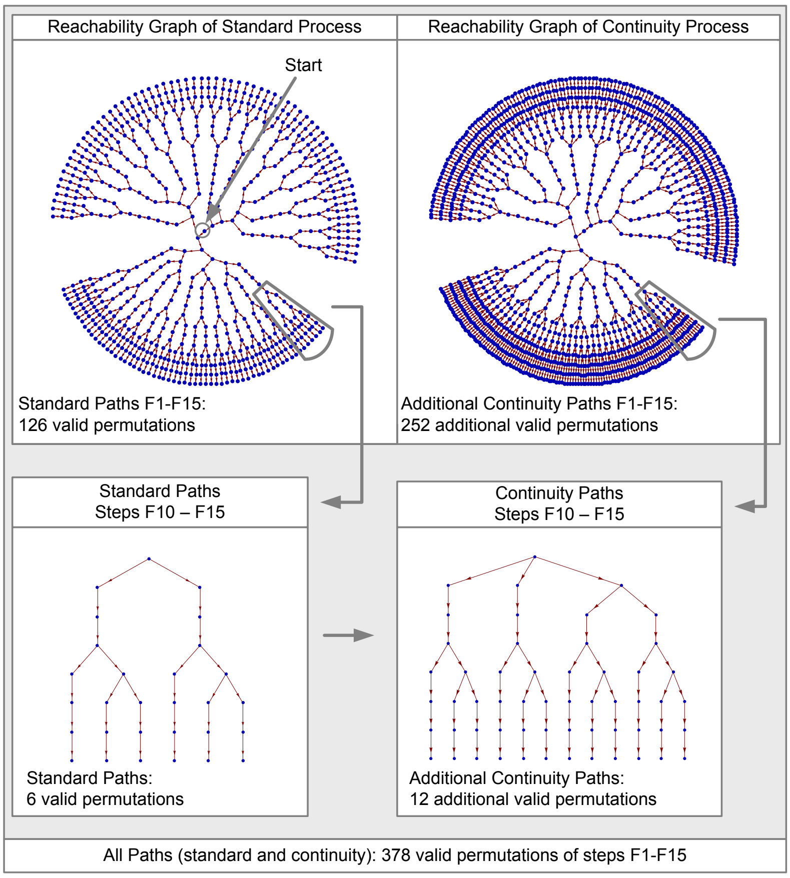

Figure 18: Universe of Continutity Paths

The first graph in Fig. 18 shows the possible paths of the standard process in Fig. 2, where the middle node represents the starting point. Each arrow in the graph represents one step. At first, there are two choices for executing a function, either the relationship manager records the customer identity or the customer demand. In both cases the next step can be to record the other data or to proceed with the storage of the already recorded data. This flexibility is not directly present in the WDEPC-model, but leaving out the syntactic events that fix the order of the steps we can derive this user friendly flexibility.

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

On the right hand side of Fig. 18 there are graphs showing the additional continuity process paths that are possible using one or more continuity snippets. For simplicity reasons the depicted graph shows the paths using snippets for the functions "RM Signature (F10)" and "Approve Contract (F12)" only, where both actors may be not available and are replaced using some of the continuity snippets given in [14]. These functions have to respect the four-eye principle as discussed at the end of Sec. 6.4 before. Indeed, there is a combination of snippets in which both actors are replaced by the credit advisor. For this reason, the generation checks the security constraint as shown in Fig. 16 on-the-fly and provides only the valid sequences. The amount of additional continuity paths is 252 and for the last 6 functions of the workflow part there are 12 additional paths.

There are certain advantages having all variants of the business process and its corresponding continuity processes generated. Firstly, in case of an emergency the Credit Suisse management can look at different options and make informed decisions which is not possible by today. This leads to some sort of recommender service. Secondly, by having all continuity processes generated continuity plans at Credit Suisse can be checked regarding their effectiveness and efficiency leading to better optimized continuity plans. We can image to use monte-carlo simulations regarding the failures in a business process given its corresponding continuity snippets. Thirdly, given the different snippets and side-constraints the universe of process variants and corresponding continuity processes can be generated beforehand making it possible for a workflow engine to react without any delay towards upcoming failures. So, this approach is compatible with real-time requirements that are showing off in certain financial transactions at Credit Suisse. Fourthly, the business process model and its corresponding continuity snippets can be stored separately, which reduces the complexity of process models and and supports ideally the decentralized modeling workflows at Credit Suisse. It further makes the administration of enterprise models easier for the people.

Summing up, once a business process is modeled its corresponding graph grammar can be derived automatically, and graph constraint checks can be performed on the abstract syntax of such a model to ensure structural security requirements. Therefore, it can be proven that a certain security requirement is valid for a certain model. By adding snippets of continuity procedures, continuity processes can be generated. Technically, this is done using process composition based on algebraic graph transformation. Continuity processes can be created in general for all possible combinations of failures from the point of view of a workflow scheme, or on a case-by-case basis from the point of view of a running workflow instance. For every generated process alternative its satisfiability is checked. The positively evaluated process schemes can be used to simulate all kinds of failures and corresponding consequences of a process instance in terms of time, operational costs and financial losses. By doing this we are able to discuss risks from a methodological point of view, not only based on organizational best-practices. Therefore, we can make informed decisions about alternatives that are fully or partially respecting the side constraints regarding security, risk and compliance. Knowing all possible continuity processes for a given critical business process we can simulate BCM risks.

After having validated these objectives, we are able to make a statement about the effectiveness and efficiency of a business continuity management system, as well as if it is economically sound in respect to security, risk and compliance requirements.

8 Transformation to *mCRL2*

In this section we explain the process modelling language *mCRL2*, show how business processes can be translated to it using graph transformation rules, and formulate a number of information flow requirements in modal logic that we have checked on the model.

8.1 Introduction

The process modelling language *mCRL2* [15] is the successor of μCRL. The letters CRL stand for Common Representation Language and the μ, which later became m is used to indicate that it is kept as concise as possible, only containing the most essential features to model discrete processes. Development on μCRL started in 1990 and work on *mCRL2* began in appr. 2003. The largest difference between the two languages is that *mCRL2* has full fledged data types (including sets, functions, quantification, etc.) whereas μCRL only offers an equational data type definition mechanism.

The essential concept in all discrete process modelling formalisms is an *action* which indicates that something happens. Actions are assumed to have no duration. They are said to be atomic in the sense that they happen atomically in time. Actions are abstractly indicated as a, b, $c, \ldots$, and in more concrete situations refered to by more verbose strings like *send*, *receive* and *drop*. Two actions can take place at the same time. This is denoted by an multi-action $a|b$. In this case it is said that they synchronize. By enforcing actions to synchronize they can be forced to communicate. For example, a *send* and *receive* can be forced to synchronize causing a communication.

There are a few operators that can be used to combine actions into behaviour. The sequential composition of processes p and q is denoted by $p{\cdot}q$ and indicates that first the behaviour of p happens, and when p terminates the behaviour of q can take place. The alternative composition $p{+}q$ indicates that either the behaviour of p or the behaviour of q can happen. The choice between the two is determined by the first action that happens in one of the two processes. So, $a{\cdot}b + c{\cdot}d$ is the process that can either do a followed by b, or c followed by d. There is one special process, called deadlock of delta, and denoted as δ which is the process that cannot do anything. It is used to indicate that behaviour will not proceed. E.g., $a{\cdot}\delta$ indicates the process that can do an action a and nothing more.

Using process equations recursive behaviour can be described. An equation $X = a{\cdot}X$ defines a process X that can do a repeatedly. Simililarly, the process equation $Y = (a{+}b){\cdot}Y$ is the process that can repeatedly and ad infinitum do an a or a b.

The parallel composition $p\|q$ is used to put processes in parallel. Because actions are atomic, they can either take place consecutively or at the same time, but they can never partly overlap. This form of parallelism is called interleaving. In an equation this is expresses as $a\|b = a{\cdot}b{+}b{\cdot}a{+}amidb$.

Actions and processes can carry any number of data parameters. Typically, one can define a process that counts by $C(n{:}\mathbb{N}) = \mathit{count}(n){\cdot}C(n{+}1)$. By the if-then-else construct $\mathit{cond}{\rightarrow}p{\diamond}q$ a condition on data *cond* can be used to select between processes p and q. The else branch can be omitted at will. Using the sum operatator a possibly infinite choice between processes can be denoted. For instance $\sum_{n:\mathbb{N}} p(n)$ represents $p(0) + p(1) + p(2) + \cdots$. The sum operator ranges over a data types, and not over a set of elements or a condition. So, in order to have a choice of all elements smaller than, say, 100, one writes $\mathit{sum}_{n:\mathbb{N}}\,(n{<}100){\rightarrow}p(n)$.

Standard data types such as $\mathbb{B}$, $\mathbb{N}$, $\mathbb{R}$ are present in the language. The faithfully represent their mathematical counterpart. E.g., there is no largest natural number. For all data types, it is possible to put them in lists, bags or sets. The sort of sets of natural numbers is denoted by $\mathit{Set}(\mathbb{N})$. Typical sets of natural number are the empty set $\{\}$, a finite set $\{1, 3, 5, 8\}$ or an infinite set $\{n{:}\mathbb{N} \mid n{>}34\}$. Of particular interest to us are enumerated types. These are defined as follows:

$$\textbf{sort } \mathit{Elements} = \textbf{struct } \mathit{elem}1 \mid \mathit{elem}2 \mid \mathit{elem}3 \mid \mathit{elem}4 \mid \mathit{elem}5;$$

This defines the sort *Elements* that contains the five mentioned elements. In this way more complex data structures can also be defined. For instance a domain with pairs of natural numbers is defined by

$$\textbf{sort } \mathit{Pair} = \textbf{struct } \mathit{pair}(\mathit{first}{:}\mathbb{N}, \mathit{second}{:}\mathbb{N});$$

So, in this case a pair of natural numbers is denoted as $\mathit{pair}(1, 2)$ and the functions *first* and *second* can be used to get the first and second element of each pair.

Using the keyword **map** new auxiliary functions can be defined and using the keyword **eqn** equations defining properties of these new functions are defined. The tools interpret these equations as rewrite rules, rewriting them from left to right. The data language domain is much richer, but for the exposition below this suffices.

When processes are put in parallel, the communication operator Γ_C is used to indicate which actions must communicate. E.g., $\Gamma_{\{send|receive \rightarrow comm\}}p$ indicates that all actions *send* and *receive* that happen synchronously in p must communicate to *comm*. If actions contain data, they can only communicate if their data parameters are the same. This can be used to hand over data, namely, the data that occurs in a *receive* action must match those in the *send* action. Using the allow operator ∇_H it can be indicated which actions are allowed. So, by writing $\nabla_{\{comm\}}(p)$ it is indicated that only communication actions can take place, and singular actions *send* and *receive* cannot occur. So, using $\nabla_{\{comm\}}\Gamma_{\{send|receive \rightarrow comm\}}p$ is a typical pattern that enforces the *receive* and *send* actions to communicate.

Also regarding the process part, we only provided the most essential elements that are used below. For a fuller exposition see for instance [15] or refer to the website `www.mcrl2.org` where the tools used below can also be obtained.

8.2 Triple Rules from WDEPC to *mCRL2*

Triple rules from WDEPC to *mCRL2* declaratively describe the relationship between the human-centric business process model to the machine-centric business process model. We assume that there is a construction grammar for the WDEPC: CWDEPC. However, we do not give it explicitly here. Triple rules consist of three different components: a source component, a connection component and a target component.

The source component shows a needed match of the source model as well as additional nodes and edges that are added after a possible match. The target component does the same for the target model. The connection component links both matches. So, that they can be looked for synchronously. Double plus signs indicate that nodes or edges are added after a possible match. Node attributes are listed in the attribute section of a node.

We do not give the complete set of triple rules for the model transformation from a WDEPC model to an *mCRL2* model, but a substantial subset that will demonstrate how the mechanism works. We like to make clear that all triple rules that are listed below work on the abstract syntax of their source and target models. Therefore, the visual appearance does not need to be fully compliant with the concrete syntax of the human-centric business model and the machine-centric business process model.

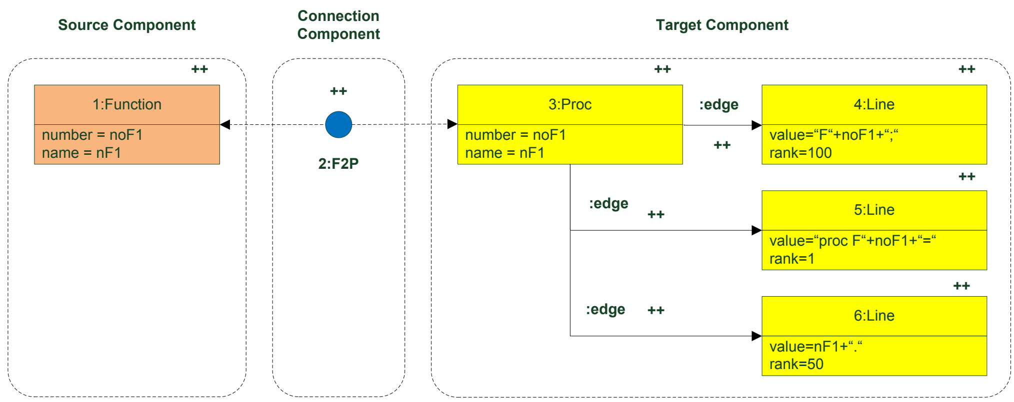

Figure 19: Triple rule: function

The triple rule "function" in Fig. 19 creates a function node (1) in the source model as well as a node for a process variable (3) in the *mCRL2* specification that are connected by a connection node (2) in the connection component. It further creates three nodes (4,5,6) representing three lines of *mCRL2* code: Two *mCRL2* actions and one definition. The definition (5) establishes the *mCRL2* code snippet for the business function, one action (6) represents a call of the business logic, the other action (4) calls the *mCRL2* process again. The mentioned ranks helps to order the lines once they are created by different triple rules.

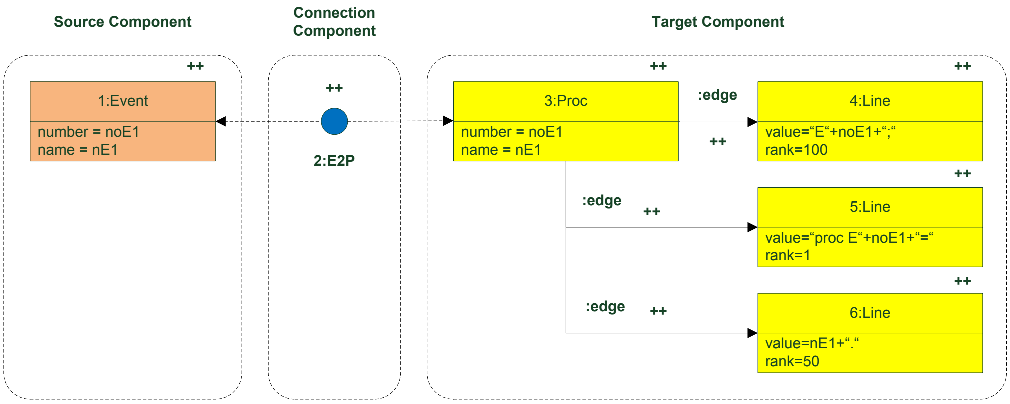

Figure 20: Triple rule: event

The triple rule "event" in Fig. 20 works like to the triple rule "function" in Fig. 19.

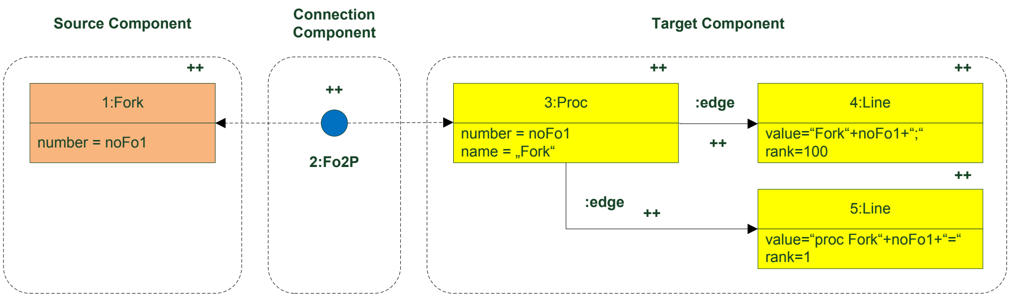

Figure 21: Triple rule: fork

The triple rule "fork" in Fig. 21 maps the fork node in the human-centric business process model into the machine-centric model. Here, only two *mCRL2* lines are created. An action as in the former triple rules is not needed because the fork node is assumed to only split the control flow without executing anything. So, there is one line defining the *mCRL2* process for this fork node (5) and one line calling this *mCRL2* process again once it was executed (4).

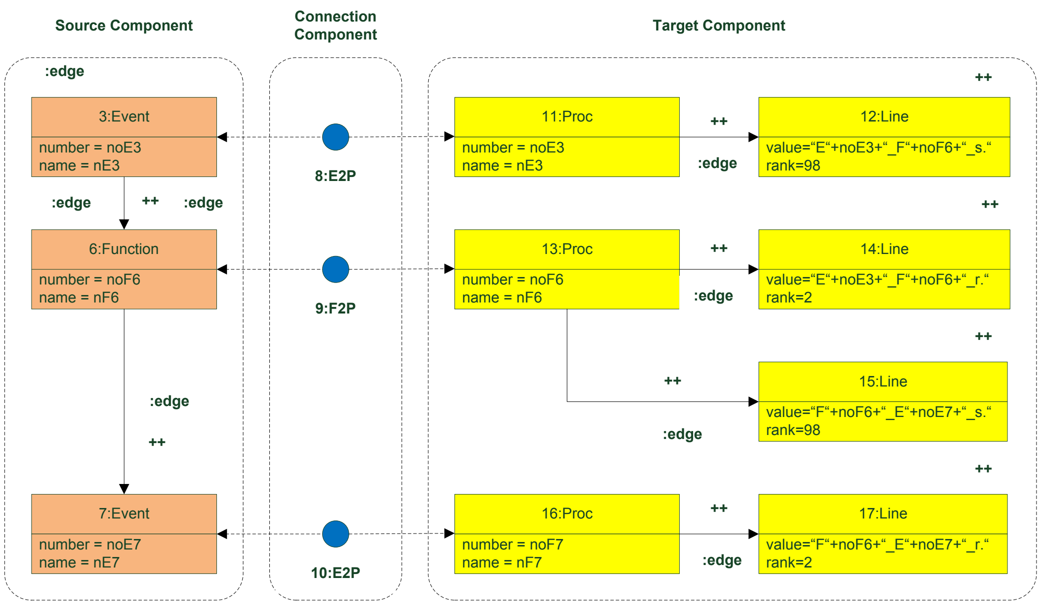

Figure 22: Triple rule: event-function-event

The triple rule “event-function-event” in Fig. 22 adds two edges in the source model to connect an event with a function and this function with another event. Nodes 12 and 14 as well as 15 and 17 in the target model represent atomic pairs of *mCRL2* actions which means that they are assumed to happen simultaneously. Therefore, the *mCRL2* code for the events and the function can be connected in a way that implements the control flow of the human-centric business process model.

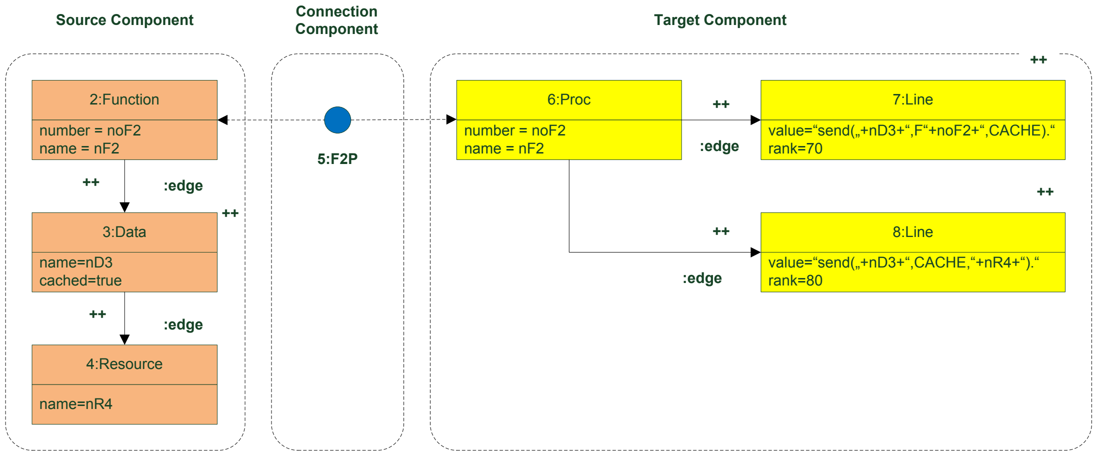

Figure 23: Triple rule: function-data-storage1

The triple rule “function-data-storage1” in Fig. 23 adds a node representing a data item into the source model that is send from a business function to a resource. It translates this by adding two lines in the *mCRL2* model. The first line (7) sends the data item from the business function to the cache, the second line sends it from the cache to the resource (8).

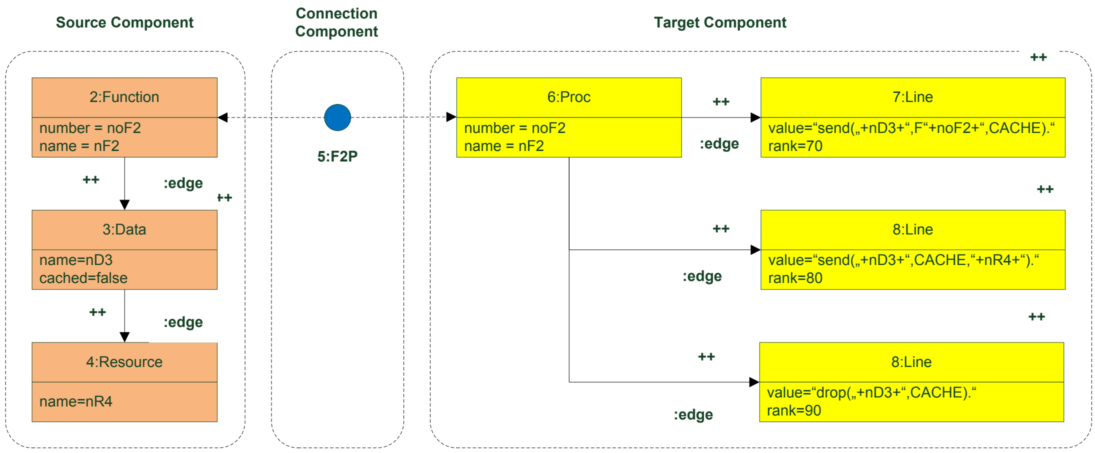

Figure 24: Triple rule: function-data-storage2

The triple rule “function-data-storage2” in Fig. 24 is very much like for former rule in Fig. 23. The only difference here is that the data is dropped by the cache. Therefore, there is a third *mCRL2* line inserted which defines that the data is dropped (8).

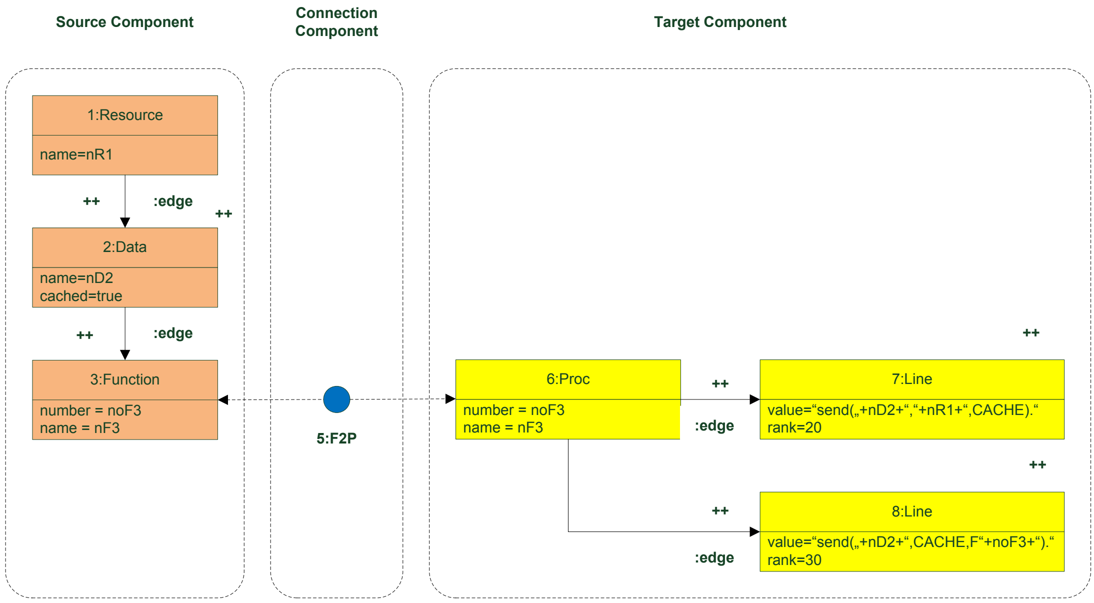

Figure 25: Triple rule: storage-data-function1

The triple rule "storage-data-function" in Fig. 25 maps the data flow between a resource and a business function in the human-centric business model to the two *mCRL2* lines in the machine-centric business process model. The first *mCRL2* line defines that the data item is send from the resource to the cache (7), the second data item defines that the data item is send from the cache to the business function (8).

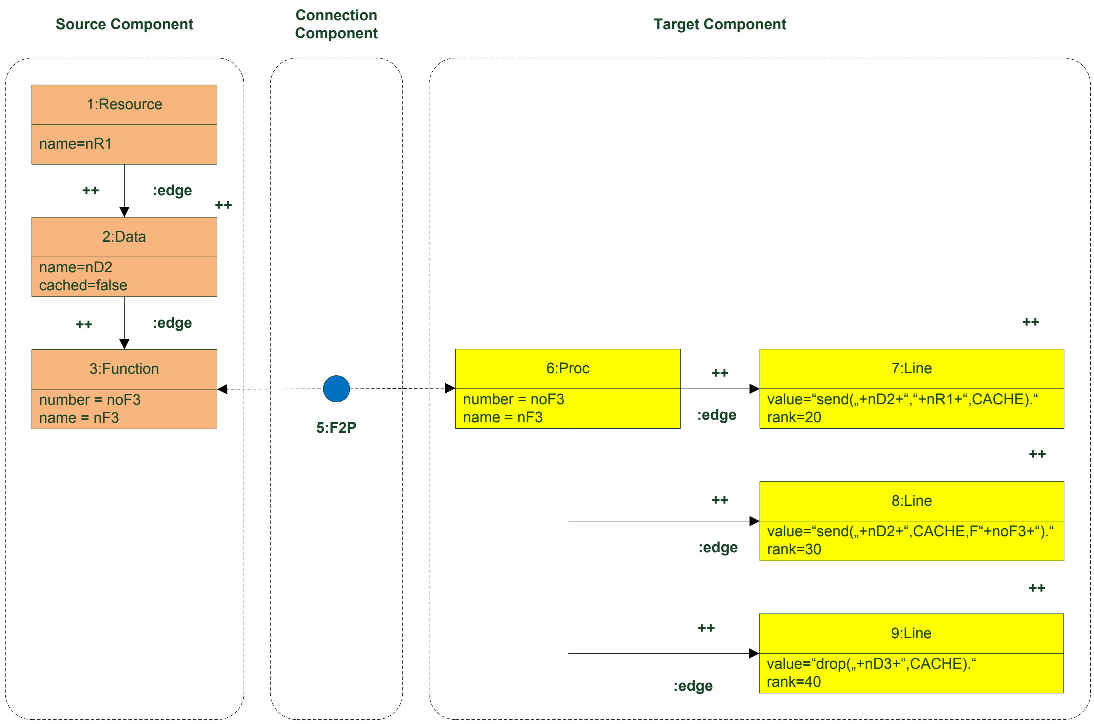

Figure 26: Triple rule: storage-data-function2

The triple rule "storage-data-function2" in Fig. 26 is similar to the former triple rule in Fig. 25 except that the data item in the cache is being dropped once the data transfer between the resource and the function is completed.

The triple rule "function-data1" in Fig. 27 maps a data transfer from a business function into

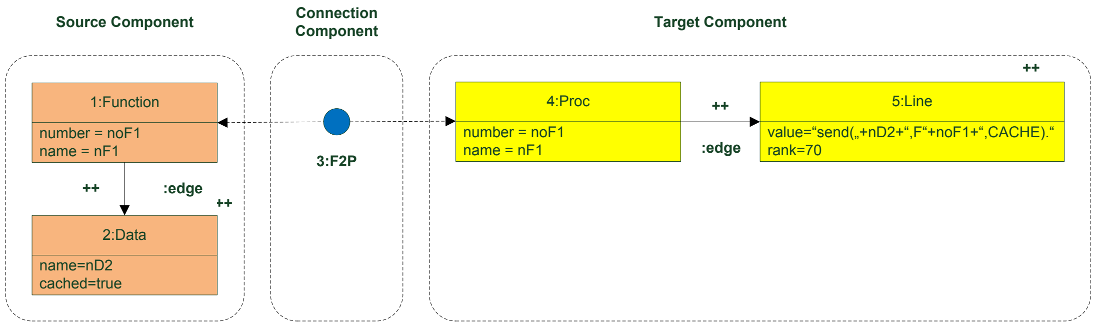

Figure 27: Triple rule: function-data1

a cache without and resource that will take the data item finally. At the *mCRL2* side there is just one line needed to implement this (5).

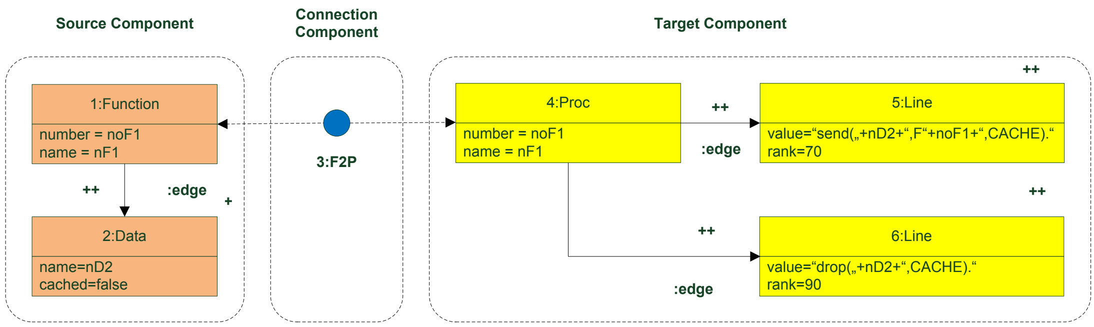

Figure 28: Triple rule: function-data2

The triple rule "function-data2" in Fig. 28 is similar to the former triple rule in Fig. 27 except that the data item in the cache is being dropped once the data transfer between the business function and the cache is completed.

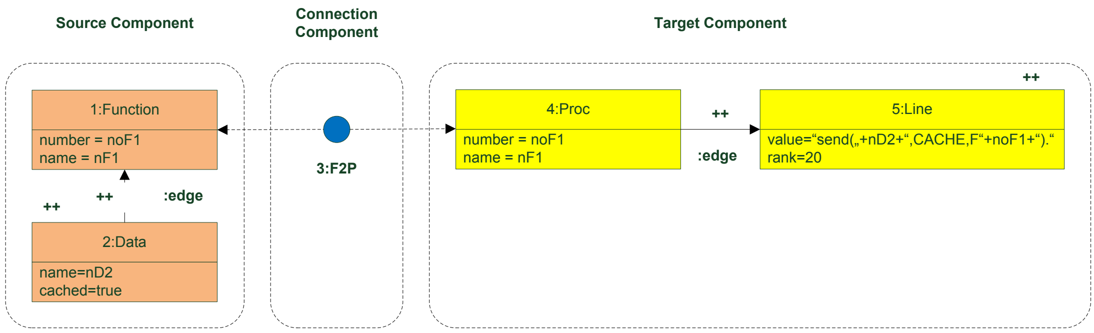

Figure 29: Triple rule: data-function1

The triple rule "data-function1" in Fig. 29 maps a data flow from a cache to a business function in the human-centric business process model to the machine-centric business process model. There it is implemented by the help of one line of *mCRL2* code (5).

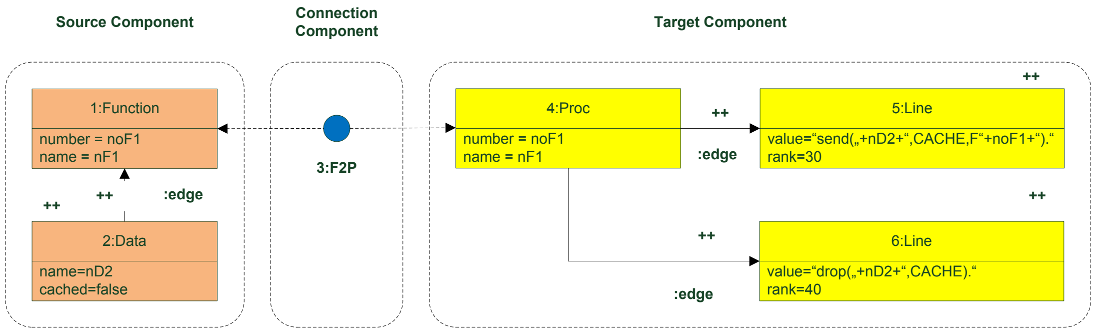

Figure 30: Triple rule: data-function2

The triple rule "data-function2" in Fig. 30 maps the data flow of a data item from the cache to a business function where the data item is not cached afterwards. It is realized by the help of two lines of *mCRL2* code in the target model (5,6). The first line transfers the data item (5), the second line (6) frees the cache.

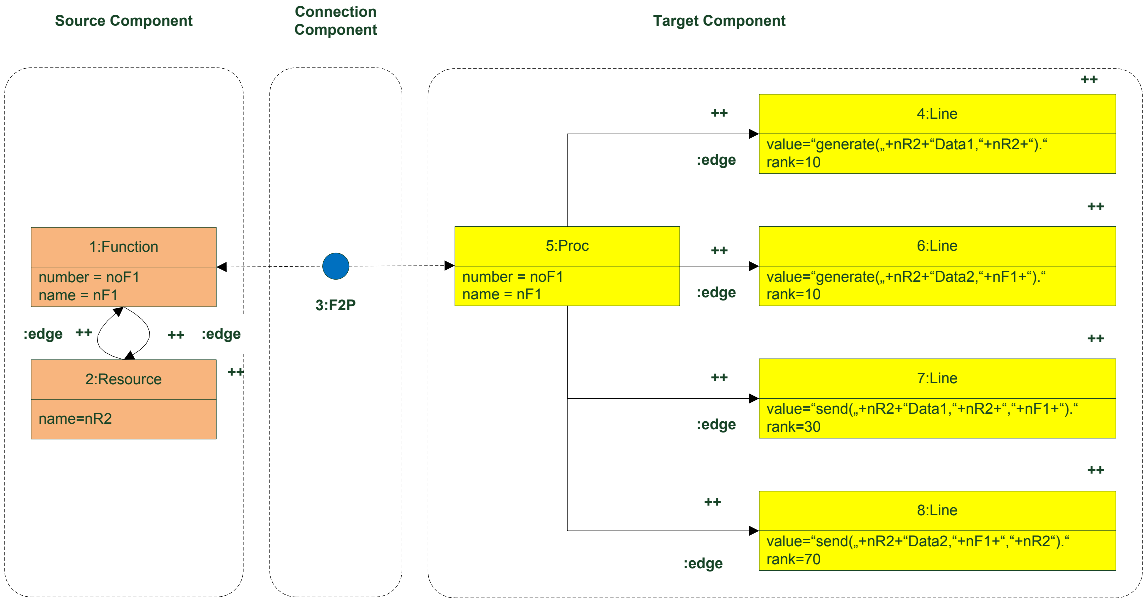

Figure 31: Triple rule: function-resource-function

The triple rule "function-resource-function" in Fig. 31 maps an anonymous data flow between a resource and a business function in both directions. In detail, two data items are generated (4,6), first the data flow from the resource to the business function is specified in *mCRL2* code (7), then the data flow from the business function to the resource is realized (8).

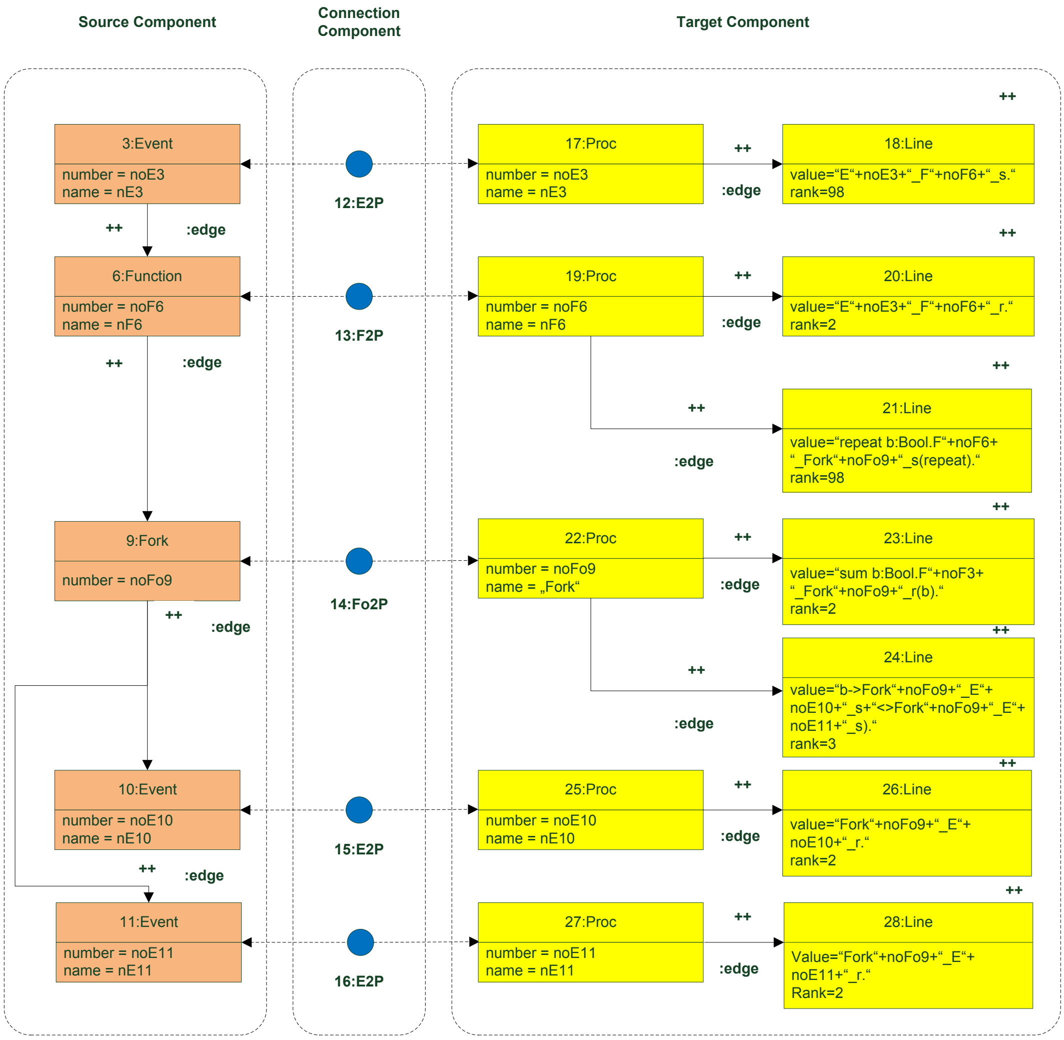

Figure 32: Triple Rule: Event-Function-Fork-Event-Event

The triple rule "event-function-fork-event-event" in Fig. 32 maps the control flow between a couple of event nodes, a function node and a fork node into the corresponding *mCRL2* specification. In detail, this is done be send and receive action pairs on the side of the *mCRL2* code that are executed as atomic actions. Therefore, they synchronize the execution of process instances. Line 23 and 24 implement the two options that are introduced by the help of the fork node in the human-centric business process model to be able to iterate over them for model checking purposes.

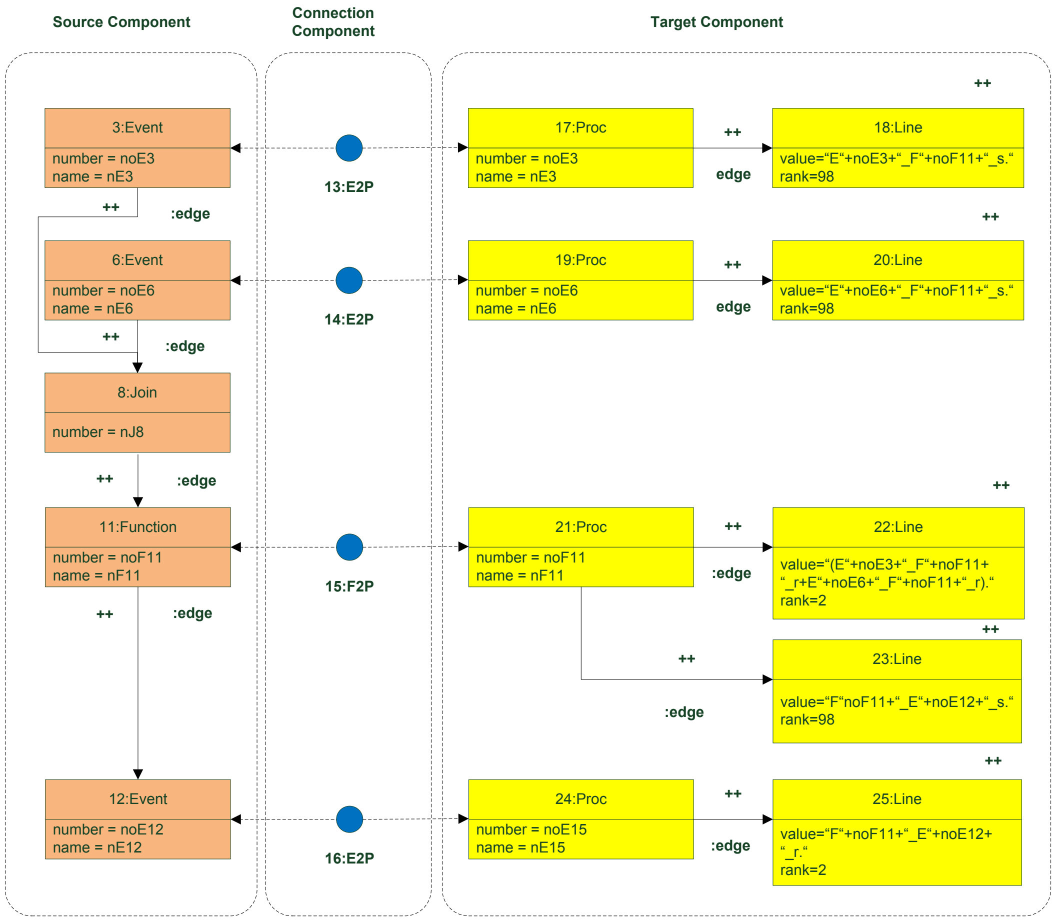

Figure 33: Triple rule: event-event-join-function-event

The triple rule "event-event-join-function-event" in Fig. 33 shows as in Fig. 32 how a control flow can be handled. Here, it is first joined before entered into the business function. The corresponding *mCRL2* code realized this control flow pattern by the help of an or-statement in line 22.

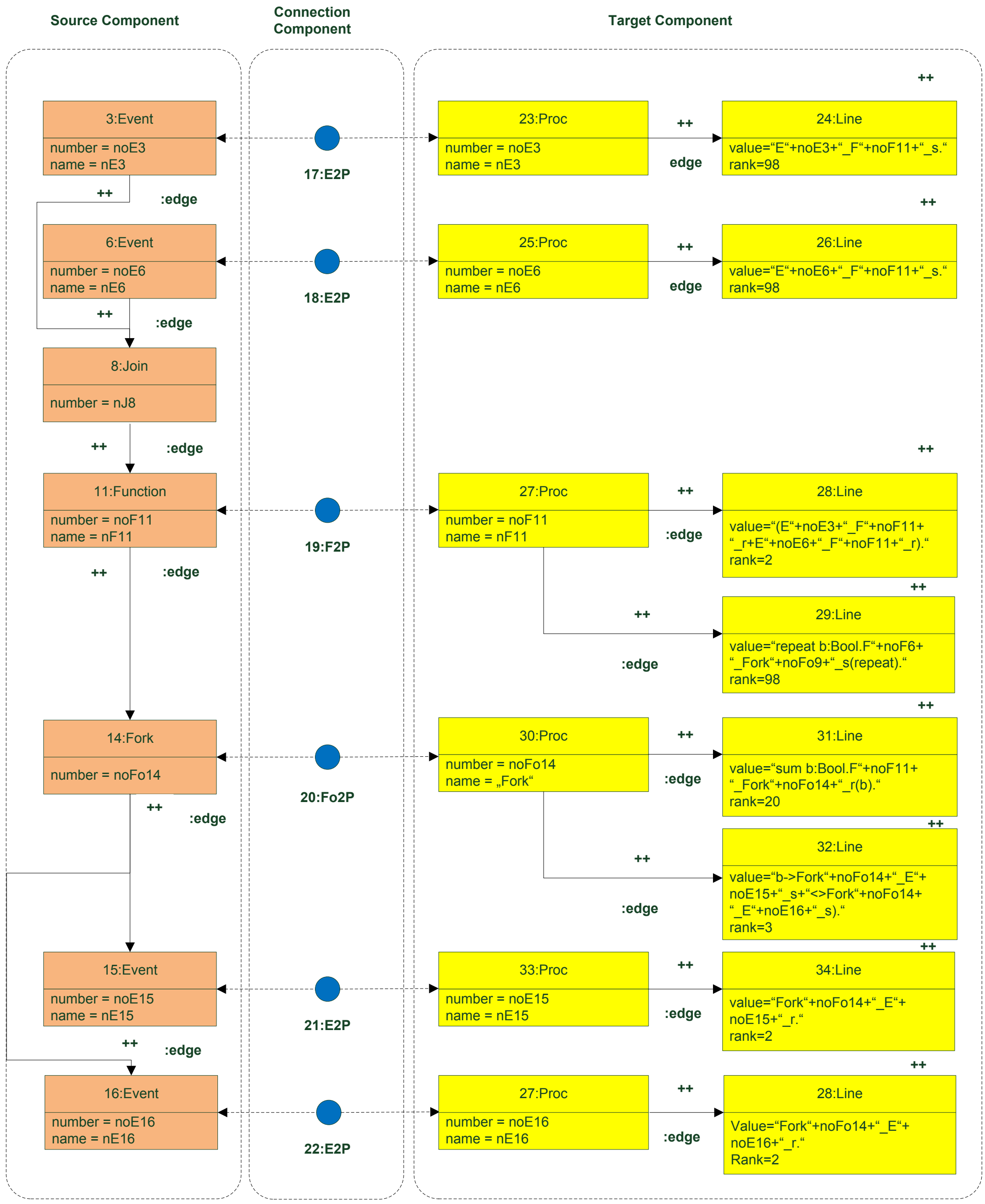

Figure 34: Triple rule: event-event-join-function-fork-event-event

The triple rule "event-event-join-function-fork-event-event" in Fig. 34 finally has a fork and a join node at the human-centric business process model side. This control flow pattern is mapped towards the machine-centric business process model side by combining the techniques introduced in Fig. 32 and Fig. 33.

8.3 Model Transformation in the Tool AGG

In order to present the effectiveness of the model transformation we present how the triple rules are used to derive the operational forward rules and how they can be executed using the graph transformation engine AGG [16]. The following figures show parts of the flattened triple graph grammar and the example in the tool AGG. The tool performes flat graph transformations, i.e. uses single graphs and not triple graphs. The corresponding flat graph grammar of our triple graph grammar can be derived as presented in [17]. Figure 35 shows the derived forward rule "NonCachedData2SendDrop". This rule translates the generated data by the function node "1" into a "send" and a "drop" operation in the *mCRL2* model.

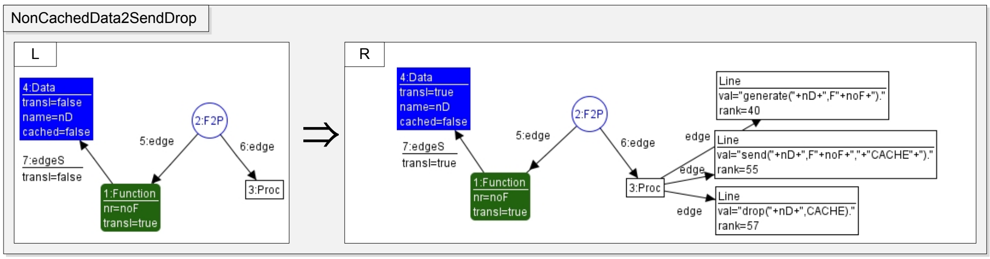

Figure 35: Model Transformation WDEPC2MCRL2: rule "NonCachedData2SendDrop" in AGG

The graph in Fig. 36 specifies a fragment of the abstract syntax graph of the WDEPC model in Fig. 2 in Sec. 4. We use this reduced example in order to show how the engine AGG creates the target model, from which the *mCRL2* code can be derived directly.

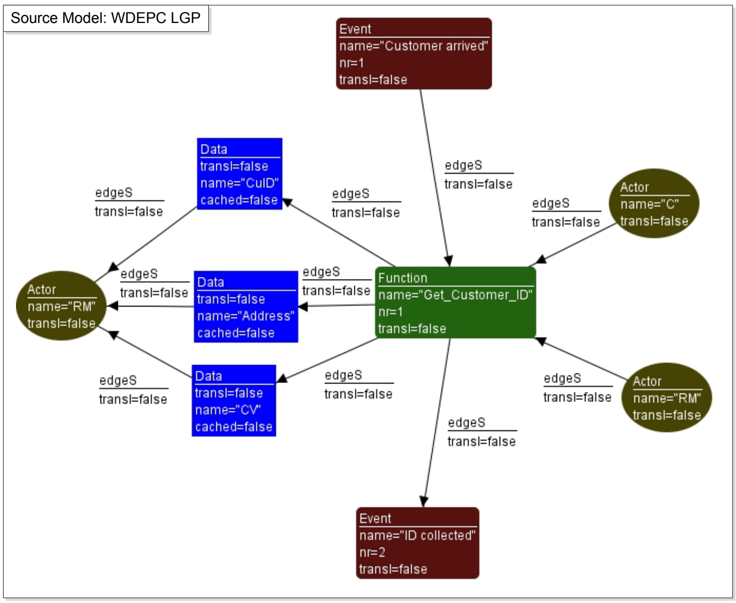

Figure 36: Model Transformation WDEPC2MCRL2: part of source model in AGG

The graph transformation engine AGG applies the derived forward rules and computes the integrated model containing the given WDEPC source model of Fig. 36 and its corresponding *mCRL2* target model fragment. This resulting integrated graph is shown in Fig. 37 and the pure target model is obtained by a restriction to the types of the target language.

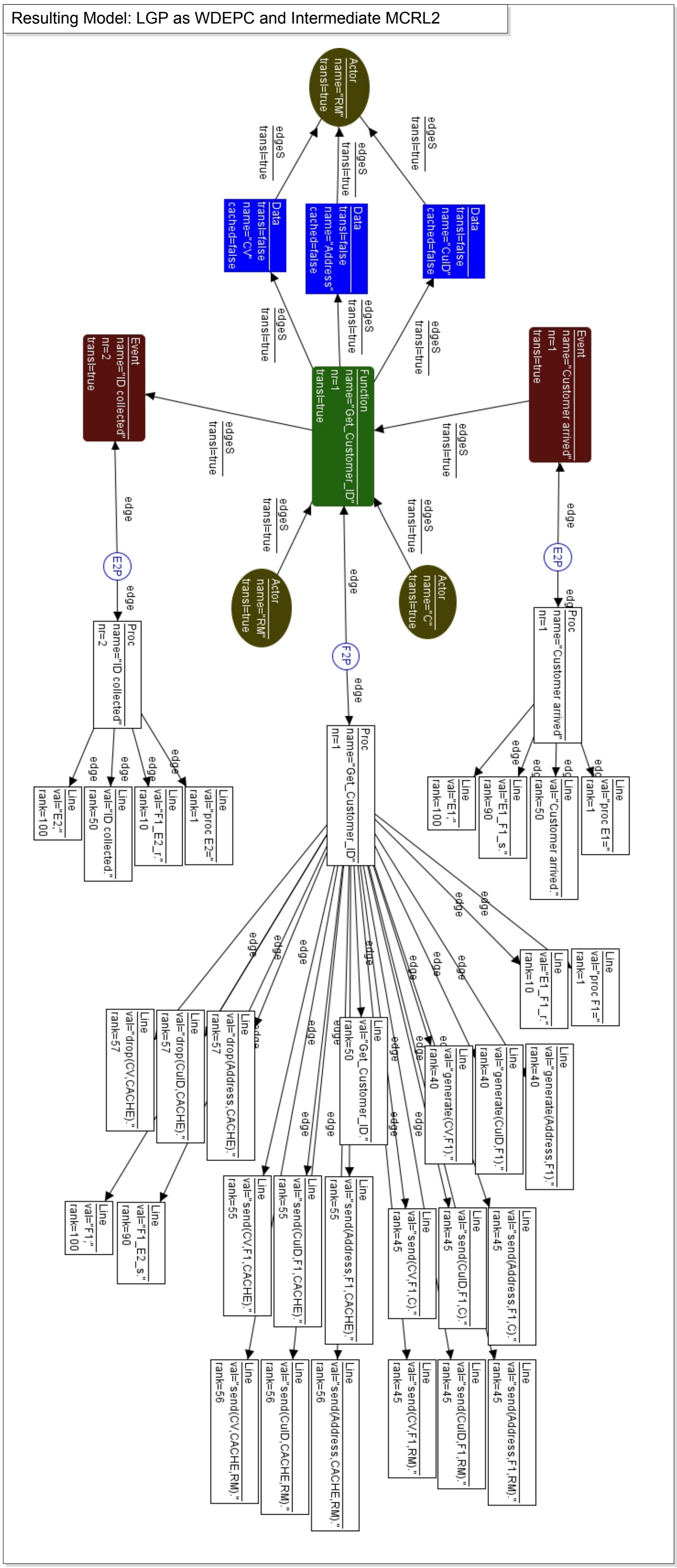

Figure 37: Model Transformation WDEPC2MCRL2: resulting integrated model in AGG

8.4 Verification of some Modal Properties

In Section 8.5 the resulting *mCRL2* code of the translation of triple graphs is given. Here we show that certain properties of this code can be verified using modal logic and the *mCRL2* toolset. First some properties are given (which are all valid), and in Section 8.4.4 it is explained how the verification will take place.

8.4.1 A data-flow requirement

The first formula expresses that the credit advisor *CA* cannot know the Address. The credit advisor can only know the address, if it is sent to him, or he generates the address himself. More precisely, there is no $send(Address, s, CA)$ action after an arbitrary sequence of actions (denoted by $true^\star$), nor is there a $generate(Address, CA)$ action that can take place after an arbitrary sequence of actions. The structure of the formula $[\ldots]\mathit{false}$ says that if one of the sequences of actions at ... exists, then one ends up in a state where *false* holds and such a state does not exist. So, such a sequence cannot occur.

```
[true*.exists s:Store.send(Address,s,CA)]false &&
[true*.generate(Address,CA))]false
```

8.4.2 An information-flow requirement

The second requirement expresses that the credit advisor *CA* cannot derive the address from the information that it has at its disposal. The data types used in this requirement are formulated at the beginning of the specification found in Section 8.5.

The modal formula is formulated as a largest fixpoint of the form $\nu X(\ldots).\phi$. At ... there is one parameter which is updated while the formula checks the behaviour. In this case the parameters is called *knowledge_set* of sort $Set(LocatedKnowledge)$ which is initially equal to the empty set ($\{\}$). The sort *LocatedKnowledge* contains pairs of a *DataString* containing some piece of data, such as an address, and a *Store*, i.e., a place where information can be stored, which can be the mind of a person but also a database. Such a pair represents that the datastring is known at that specific location. So, *knowledge_set* is a set of all data strings that are known at specific spots.

The second line of the formula contains the check that the address is not known by the credit advisor, or in other words, that $pair(Address, CA)$ is not an element of the *knowledge_set*.

The other lines of the formula say that the *knowledge_set* is properly maintained. So, whenever a $generate(d, s)$ action take place for any data string d, and store s (denoted by $[generate(d, s)]$ using square brackets), then the data string d is known by location s, and this pair must be added to the *knowledge_set*. Furthermore, all derivable information must also be added to those places where it is derivable. This is done by the function *AddInfo* defined in the specification in Section 8.5. Using the function *InfoFlow* which says how information flows from data string to data string, *AddInfo* puts all data strings in those locations where they can be inferred.

Similarly, when a data string is sent from store s_1 to s_2 by $send(d, s_1, s_2)$ then using the function *AddInfo* it is indicated where data can become known (line 5). In line 4 it is indicated that when data is dropped from a store, it can still be remembered, and therefore, it is not removed from the knowledge set. Likewise, in lines 6-8 it is indicated that other actions than *generate*, *drop* and *send* do not influence *knowledge_set* either.

```
1 nu X(knowledge_set:Set(LocatedKnowledge)={}).
2  !(val(pair(Address,CA) in knowledge_set)) &&
3  (forall d:DataString,s:Store.[generate(d,s)] X(AddInfo(d,s,knowledge_set))) &&
4  (forall d:DataString,s:Store.[drop(d,s)] X(knowledge_set)) &&
5  (forall d:DataString,s1,s2:Store.[send(d,s1,s2)] X(AddInfo(d,s2,knowledge_set))) &&
6  [(forall d:DataString,s:Store.!generate(d,s)) &&
7    (forall d:DataString,s:Store.!drop(d,s)) &&
8    (forall d:DataString,s1,s2:Store.!send(d,s1,s2))]X(knowledge_set)
```

8.4.3 An information-derivation requirement

Besides tracking the flow of information, one can also assume that different pieces of information can be obtained from which some information can be derived. E.g., if one knows the customer demand (CD), one might know the address with a certainty of $\frac{1}{20}$, and the credit worthiness CW

with a certainty of $\frac{9}{10}$. This is formulated in the function *QuantInfoFlow* in the specification in Section 8.5.

The formula below says that the resource manager *RM* will be able to derive the rating for the customer with more that $\frac{7}{10}$ certainty. We adopt the rule that if information can be derived from two sources, one with certainty p_1 and the other with certainty p_2, then we assume that the information is known with certainty $1-(1-p_1)(1-p_2)$ (see line 77 in the specification in Section 8.5.

Basically, the formula below maintains what is known by which store in *knowledge_list*. At line 2 and 3 it is checked whether the resource manager knows the rating with more than $\frac{7}{10}$ probability. The function *SelectInformation* selects the information for a particular store (in this case *RM*) with certainty 1. The function *QuantInferredInformation* is used to calculate the probabilities with which the data can be derived from the information known to the resource manager. The function *GetQuantInfo* selects the probability with which the rating is known. These functions are defined in the specification in Section 8.5.

```
1 nu X(knowledge_list:List(LocatedKnowledge)=[]).
2  (val(GetQuantInfo(Rating,QuantInferredInformation(SelectInformation(RM,knowledge_list),
3                                SelectInformation(RM,knowledge_list)))>7/10)) &&
4  (forall d:DataString,s:Store.[generate(d,s)] X(Insert(d,s,knowledge_list))) &&
5  (forall d:DataString,s:Store.[drop(d,s)] X(knowledge_list)) &&
6  (forall d:DataString,s1,s2:Store.[send(d,s1,s2)] X(Insert(d,s2,knowledge_list))) &&
7  [(forall d:DataString,s:Store.!generate(d,s)) &&
8   (forall d:DataString,s:Store.!drop(d,s)) &&
9   (forall d:DataString,s1,s2:Store.!send(d,s1,s2))]X(knowledge_list)
```

8.4.4 Verification commands

Below the commands are given to verify a modal formula that is put in a file `modal_formula.mcf`. The specification is put in the file *LendingProcess2.mcrl2*. The idea behind the toolset for *mCRL2* is that there are a number of separate tools to transform a specification. Depending on the nature of the specification and the properties that must be checked, a different sequence of tools can be invoked. It goes too far to explain all the tools that have been used here (see for instance `www.mcrl2.org` where they are all explained).

The most important tools below are `mcrl22lps` that translates the specification in Section 8.5 to a linear process which is essentially a set of condition-action-effect rules. A linear process is a very simple basic process form, behaviourally equivalent to the original, where process structuring operators, such as the parallel operator have been eliminated. Using the `lps2pbes` tool the linear process together with a modal formula is translated to a parameterised boolean equation system (PBES, [18]). A PBES is essentially a sequence of fixed point equation which must be solved. The solution of the PBES indicates whether the modal formula is valid for the specification or not. Line 6 indicates how to generate a labelled transition system but it is not used for the verification of a modal formula.

```
1 mcrl22lps -vD LendingProcess2.mcrl2 temp.lps
2 lpssuminst -v temp.lps | lpsrewr | lpsconstelm -v | lpsrewr > temp1.lps
3 lps2pbes -v -f modal_formula.mcf temp1.lps temp.pbes
4 pbesrewr -v -pquantifier-finite -rjittyc temp.pbes temp1.pbes
5 pbes2bool -s2 -v temp1.pbes
6 lps2lts temp1.lps temp.aut
```

8.5 Model for Lending Process

This section contains the translation of the workflow shown in Figure 2. The part up till the description of the datatypes contains auxiliary data type definitions for the modal formulas. These data types have shortly been explained above.

The main data types for the specification (line 104 and line 109) contain the data strings `DataString` and the places where information can be stored (`Store`).

The subsequent part of the specification (line 120 till line 548) contain the translated processes. E.g. process *E1* indicates that a customer arrives, using a *Customer_arrived* event. Subsequently, it sends a trigger to process *F1* that it can proceed using the event *E1_F1_s*. In process *F1* the trigger is received using the event *E1_F1_r*. In the `comm` command at line 668 it is defined how such events must communicate. Subsequently, in *F1* information is some information is generated, which is modelled using the *generate* event. The event *Get_customer_identity* takes place and subsequently, there are events indicating to which 'stores' the information is sent. All processes, except *E1* are repeated at the end. In *E1*, the process ends in `delta` to prevent more than one customer from entering the system. This suffices, as we want to investigate the information flow regarding only one customer.

At the end, all processes and events are put in parallel. Using the `comm` and `allow` operator it is prescribed how events must communicate, and which events are externally visible (others are blocked). In the actual verification, we had to apply alphabet axioms to push the allow operator inside the processes, as otherwise the `mcrl22lps` tool would require too much time. We left this out, as otherwise the init section at the end would become much larger.

```
map  InfoFlow:DataString->List(DataString);
eqn  InfoFlow(CuID)=[CuID];
     InfoFlow(Address)=[Address,CV,CW,Rating];
     InfoFlow(CV)=[CV,CW,Rating];
     InfoFlow(CD)=[Address,CD,CV,CW,Product,Rating];
     InfoFlow(CW)=[CV,CW,Rating];
     InfoFlow(Rating)=[CV,CW,Rating];
     InfoFlow(PP)=[PP];
     InfoFlow(Product)=[Address,CD,CV,CW,Product,Rating];
     InfoFlow(CoID)=[ClosingDate,CoID];
     InfoFlow(SRM_RMID)=[SRM_RMID];
     InfoFlow(SC)=[SC];
     InfoFlow(SCO_COID)=[SCO_COID];
     InfoFlow(MT1)=[CV,CW,MT1,Rating];
     InfoFlow(MT2)=[CV,CW,MT2,Rating];
     InfoFlow(SchufaData1)=[Address,CD,CV,CW,Product,Rating,SchufaData1];
     InfoFlow(SchufaData2)=[SchufaData2];
     InfoFlow(ClosingDate)=[ClosingDate];

map  AddInfo:DataString#Store#Set(LocatedKnowledge)->Set(LocatedKnowledge);
     AddInfoList:DataString#Store#Set(LocatedKnowledge)#List(DataString)->
                                                            Set(LocatedKnowledge);

var  d,d':DataString;
     s:Store;
     set:Set(LocatedKnowledge);
     l:List(DataString);
eqn  AddInfo(d,s,set)=AddInfoList(d,s,set,InfoFlow(d));
     AddInfoList(d,s,set,[])=set;
     AddInfoList(d,s,set,d'|>l)=set+{pair(d',s)};

% The function QuantInfFlow (Quantitative Information Flow) shows how
% quantitative information flows from one DataString to another. The function
% indicates the direct information flow. The indirect flow is excluded. This must
% be done, as indirect information can come via different dependent ways, and
% should not be added in that case.

sort InformationPair=struct pair(data:DataString,quantity:Real);
     LocatedKnowledge= struct pair(data:DataString,store:Store);

map  QuantInfoFlow:DataString->List(InformationPair);
eqn  QuantInfoFlow(CuID)=[];
     QuantInfoFlow(Address)=[pair(CV,1/2)];
     QuantInfoFlow(CV)=[pair(CW,1/2),pair(Rating,9/10)];
     QuantInfoFlow(CD)=[pair(Address,1/20),pair(Product,2/3)];
     QuantInfoFlow(CW)=[pair(Rating,9/10)];
     QuantInfoFlow(Rating)=[pair(CV,8/9)];
     QuantInfoFlow(PP)=[];
     QuantInfoFlow(Product)=[pair(CD,1/2)];
     QuantInfoFlow(CoID)=[pair(ClosingDate,2/3)];
     QuantInfoFlow(SRM_RMID)=[];
     QuantInfoFlow(SC)=[];
     QuantInfoFlow(SCO_COID)=[];
     QuantInfoFlow(MT1)=[pair(Rating,1/10)];
     QuantInfoFlow(MT2)=[pair(Rating,2/5)];
     QuantInfoFlow(SchufaData1)=[pair(Rating,1/2)];
     QuantInfoFlow(SchufaData2)=[];
     QuantInfoFlow(ClosingDate)=[];

```

```
map  QuantInferredInformation,Combine:List(InformationPair)#List(InformationPair)->
                                                    List(InformationPair);
     DirectDerivatives:List(InformationPair)->List(InformationPair);
     Weight:Real#List(InformationPair)->List(InformationPair);
     Insert:DataString#Store#List(LocatedKnowledge)->List(LocatedKnowledge);
     GetQuantInfo:DataString#List(InformationPair)->Real;
     SelectInformation:Store#List(LocatedKnowledge)->List(InformationPair);
var  l,l':List(InformationPair);
     m:List(LocatedKnowledge);
     d,d':DataString;
     s,s':Store;
     r,r':Real;
eqn  QuantInferredInformation(l,[])=l;
     l'!=[] -> QuantInferredInformation(l,l')=Combine(l,DirectDerivatives(l'));

     Combine(l,[])=l;
     Combine([],l')=l';
     Combine(pair(d,r)|>l,pair(d,r')|>l')=pair(d,1-(1-r)*(1-r'))|>Combine(l,l');
     d<d' -> Combine(pair(d,r)|>l,pair(d',r')|>l')=pair(d,r)|>Combine(l,pair(d',r')|>l');
     d>d' -> Combine(pair(d,r)|>l,pair(d',r')|>l')=pair(d',r')|>Combine(pair(d,r)|>l,l');
     DirectDerivatives([])=[];
     DirectDerivatives(pair(d,r)|>l)=
              if(r<1/100,
                   DirectDerivatives(l),
                   Combine(Weight(r,QuantInfoFlow(d)),DirectDerivatives(l)));
     Weight(r,[])=[];
     Weight(r,pair(d,r')|>l)=pair(d,r*r')|>Weight(r,l);
     Insert(d,s,[])=[pair(d,s)];
     (d<d' || (d==d' && s<s'))-> Insert(d,s,pair(d',s')|>m)=pair(d,s)|>pair(d',s')|>m;
     Insert(d,s,pair(d,s)|>m)=pair(d,s)|>m;
     (d>d' || (d==d' && s>s'))-> Insert(d,s,pair(d',s')|>m)=pair(d',s')|>Insert(d,s,m);
     GetQuantInfo(d,[])=0;
     GetQuantInfo(d,pair(d',r)|>l)=if(d==d',r,GetQuantInfo(d,l));
     SelectInformation(s,[])=[];
     SelectInformation(s,pair(d,s')|>m)=if(s==s',pair(d,1)|>SelectInformation(s,m),
                                                   SelectInformation(s,m));


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                               %
%   Description of datatypes                                    %
%                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

sort DataString = struct CuID | Address | CV | CD |
                         CW | Rating | PP | Product | CoID |
                         SRM_RMID | SC | SCO_COID | MT1 | MT2 |
                         SchufaData1 | SchufaData2 | ClosingDate;

sort Store = struct CACHE | C | RM | CA | CO | DB | DB1 | DB2 | DB3 |
                    F1 | F2 | F3 | F4 | F5 | F6 | F7 | F8 | F9 | F10 |
                    F11 | F12 | F13 | F14 | F15 | DBSchufa | Contract;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                               %
%   Description of tasks and events                             %
%                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

proc E1=
        Customer_arrived.
        E1_F1_s.
        delta;                        % Only once customer can arrive.

proc F1=
        E1_F1_r.

        generate(CuID,F1).
        generate(Address,F1).
        generate(CV,F1).

        Get_customer_identity.
```

```

        send(CuID,F1,RM).
        send(Address,F1,RM).
        send(CV,F1,RM).
        send(Address,F1,C).
        send(CV,F1,C).
        send(CuID,F1,C).

        send(CuID,F1,CACHE).
        send(CuID,CACHE,RM).
        drop(CuID,CACHE).
        send(Address,F1,CACHE).
        send(Address,CACHE,RM).
        drop(Address,CACHE).
        send(CV,F1,CACHE).
        send(CV,CACHE,RM).
        drop(CV,CACHE).

        F1_E2_s.

        F1;

proc E2=
        F1_E2_r.

        ID_collected.

        E2_F2_s.
        E2;

proc F2=
        E2_F2_r.

        generate(CD,F2).

        Get_customer_demand.

        send(CD,F2,C).
        send(CD,F2,RM).

        send(CD,F2,CACHE).
        send(CD,CACHE,RM).
        drop(CD,CACHE).

        F2_E3_s.
        F2;

proc E3=
        F2_E3_r.

        Data_collected.

        E3_F3_s.
        E3;

proc F3=
         E3_F3_r.

         send(CuID,RM,CACHE).
         send(CuID,CACHE,F3).
         drop(CuID,CACHE).
         send(Address,RM,CACHE).
         send(Address,CACHE,F3).
         drop(Address,CACHE).
         send(CV,RM,CACHE).
         send(CV,CACHE,F3).
         drop(CV,CACHE).

         Store_ID_Data.

         send(CuID,F3,RM).
         send(Address,F3,RM).
         send(CV,F3,RM).
```

```
          send(CuID,F3,CACHE).
          send(CuID,CACHE,DB1).
          send(Address,F3,CACHE).
          send(Address,CACHE,DB1).
          send(CV,F3,CACHE).
          send(CV,CACHE,DB1).
          F3_E4_s.
          F3;

proc E4=
         F3_E4_r.

         ID_data_stored.

         E4_F4_s.
         E4;

proc F4=
         E4_F4_r.

         send(CD,RM,CACHE).
         send(CD,CACHE,F4).
         drop(CD,CACHE).

         Store_customerDemand_data.

         send(CD,F4,RM).

         send(CD,F4,CACHE).
         send(CD,CACHE,DB1).

         F4_E5_s.
         F4;

proc E5=
         F4_E5_r.

         Data_stored.

         E5_F5_s.
         E5;

proc F5=
         E5_F5_r.

         send(CuID,CACHE,F5).
         send(Address,CACHE,F5).
         generate(SchufaData1,DBSchufa).
         send(SchufaData1,DBSchufa,F5).

         generate(SchufaData2,F5).
         generate(CW,F5).

         Credit_worthiness.

         send(SchufaData2,F5,DBSchufa).
         send(CW,F5,CACHE).
         send(CW,CACHE,DB1).
         F5_E6_s.
         F5;

proc E6=
         F5_E6_r.

         CreditWorthyness_computed.

         E6_F6_s.
         E6;

proc F6=
         E6_F6_r.

```

```
        send(CuID,CACHE,F6).
        send(Address,CACHE,F6).
        send(CV,CACHE,F6).
        send(CW,CACHE,F6).

        generate(Rating,F6).

        Rating_customer.

        send(Rating,F6,CACHE).
        send(Rating,CACHE,DB1).

        F6_E7_s.
        F6;

proc E7=
        F6_E7_r.

        Rating_computed.

        E7_F7_s.
        E7;

proc F7=
        E7_F7_r.

        send(CuID,CACHE,F7).
        send(Rating,CACHE,F7).

        sum accept:Bool.Customer_acceptance.

        send(CuID,F7,CA).
        send(Rating,F7,CA).

        F7_Split1_s(accept).
        F7;

proc Split1=
        sum b:Bool.F7_Split1_r(b).
        (b->Split1_E8_s<>Split1_E9_s).
        Split1;

proc E8=
        Split1_E8_r.

        Customer_accepted.

        E8_F8_s.
        E8;

proc E9=
        Split1_E9_r.
        Customer_not_accepted.
        E9;

proc F8=
        E8_F8_r.

        send(CV,CACHE,F8).
        send(CW,CACHE,F8).
        send(CD,CACHE,F8).
        generate(Product,F8).

        Create_optimized_product.

        send(Product,F8,CA).
        send(CV,F8,CA).
        send(CW,F8,CA).
        send(CD,F8,CA).

        send(Product,F8,CACHE).
        send(Product,CACHE,DB1).

```

```
        F8_E10_s.
        F8;

proc E10=
        F8_E10_r.
        Price_computed.
        E10_F9_s.
        E10;

proc F9=
        E10_F9_r.

        send(Product,DB1,CACHE).
        send(Product,CACHE,F9).
        send(CuID,CACHE,F9).

        generate(CoID,F9).
        generate(PP,F9).

        Create_contract.

        send(Product,F9,RM).
        send(CuID,F9,RM).
        send(CoID,F9,RM).
        send(PP,F9,RM).

        send(PP,F9,CACHE).
        send(PP,CACHE,DB2).
        send(CoID,F9,CACHE).
        send(CoID,CACHE,DB2).

        send(CoID,CACHE,Contract).
        send(PP,CACHE,Contract).

        F9_E11_s.
        F9;

proc E11=
        F9_E11_r.
        Contract_created.
        E11_F10_s.
        E11;

proc F10=
        E11_F10_r.

        send(CoID,CACHE,F10).
        generate(SRM_RMID,F10).

        RM_signature.

        send(SRM_RMID,F10,RM).
        send(CoID,F10,RM).

        send(SRM_RMID,F10,CACHE).
        send(SRM_RMID,CACHE,DB2).
        send(SRM_RMID,CACHE,Contract).

        F10_E12_s.
        F10;

proc E12=
        F10_E12_r.
        RM_has_signed.
        E12_F11_s.
        E12;

proc F11=
        E12_F11_r.

        send(CoID,CACHE,F11).
        generate(SC,F11).

```

```
        Customer_signature.

        send(SC,F11,C).
        send(CoID,F11,C).

        send(SC,F11,CACHE).
        send(SC,CACHE,DB2).
        send(SC,CACHE,Contract).

        F11_E13_s.
        F11;

proc E13=
        F11_E13_r.
        Contract_signed.
        E13_F12_s.
        E13;

proc F12=
        E13_F12_r.

        send(SRM_RMID,CACHE,F12).
        send(CoID,CACHE,F12).
        generate(SCO_COID,F12).

        Approve_contract.

        send(SRM_RMID,F12,CO).
        send(SCO_COID,F12,CO).
        send(CoID,F12,CO).

        send(SCO_COID,F12,CACHE).
        send(SCO_COID,CACHE,DB2).
        send(SCO_COID,CACHE,Contract).

        F12_E14_s.
        F12;

proc E14=
        F12_E14_r.
        Contract_approved.
        E14_F13_s.
        E14;

proc F13=
        E14_F13_r.

        send(CoID,CACHE,F13).
        generate(MT1,F13).

        Cash_payment.

        send(CoID,F13,C).
        send(MT1,F13,C).

        send(MT1,F13,CACHE).
        send(MT1,CACHE,DB2).

        F13_E15_s.
        F13;

proc E15=
        F13_E15_r.
        Cash_paid.
        E15_F14_s.
        E15;

proc F14=
        (E15_F14_r+E16_F14_r).

        send(CoID,CACHE,F14).
        send(PP,CACHE,F14).
        generate(MT2,F14).
```

```

        Cash_returned.

        send(CoID,F14,C).
        send(PP,F14,C).
        send(MT2,F14,C).

        send(MT2,F14,CACHE).
        send(MT2,CACHE,DB3).
        send(PP,F14,CACHE).
        send(PP,CACHE,DB3).

        sum repeat:Bool.F14_Split2_s(repeat).
        F14;

proc Split2=
        sum b:Bool.F14_Split2_r(b).
        (b->Split2_E17_s<>Split2_E16_s).
        Split2;

proc E16=
        Split2_E16_r.
        Open_PP.
        E16_F14_s.
        E16;

proc E17=
        Split2_E17_r.
        Closed_PP.
        E17_F15_s.
        E17;

proc F15=
        E17_F15_r.

        send(CoID,CACHE,F15).
        send(MT1,CACHE,F15).
        generate(ClosingDate,F15).

        Close_contract.

        send(ClosingDate,F15,CACHE).
        send(ClosingDate,CACHE,DB3).

        F15_E18_s.
        F15;

proc E18=
        F15_E18_r.
        Contract_closed.
        E18;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                %
%   Declaration of actions                                       %
%                                                                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

act  E1_F1_s, E1_F1_r, E1_F1_c,
     F1_E2_s, F1_E2_r, F1_E2_c,
     E2_F2_s, E2_F2_r, E2_F2_c,
     F2_E3_s, F2_E3_r, F2_E3_c,
     E3_F3_s, E3_F3_r, E3_F3_c,
     F3_E4_s, F3_E4_r, F3_E4_c,
     E4_F4_s, E4_F4_r, E4_F4_c,
     F4_E5_s, F4_E5_r, F4_E5_c,
     E5_F5_s, E5_F5_r, E5_F5_c,
     F5_E6_s, F5_E6_r, F5_E6_c,
     E6_F6_s, E6_F6_r, E6_F6_c,
     F6_E7_s, F6_E7_r, F6_E7_c,
     E7_F7_s, E7_F7_r, E7_F7_c,
     Split1_E8_s, Split1_E8_r, Split1_E8_c,
```

```
     Split1_E9_s, Split1_E9_r, Split1_E9_c,
     E8_F8_s, E8_F8_r, E8_F8_c,
     F8_E10_s, F8_E10_r, F8_E10_c,
     E10_F9_s, E10_F9_r, E10_F9_c,
     F9_E11_s, F9_E11_r, F9_E11_c,
     E11_F10_s, E11_F10_r, E11_F10_c,
     F10_E12_s, F10_E12_r, F10_E12_c,
     E12_F11_s, E12_F11_r, E12_F11_c,
     F11_E13_s, F11_E13_r, F11_E13_c,
     E13_F12_s, E13_F12_r, E13_F12_c,
     F12_E14_s, F12_E14_r, F12_E14_c,
     E14_F13_s, E14_F13_r, E14_F13_c,
     F13_E15_s, F13_E15_r, F13_E15_c,
     E15_F14_s, E15_F14_r, E15_F14_c,
     E16_F14_s, E16_F14_r, E16_F14_c,
     Split2_E17_s, Split2_E16_r, Split2_E16_c,
     Split2_E16_s, Split2_E17_r, Split2_E17_c,
     E17_F15_s, E17_F15_r, E17_F15_c,
     F15_E18_s, F15_E18_r, F15_E18_c;

act  F7_Split1_s, F7_Split1_r, F7_Split1_c,
     F14_Split2_s, F14_Split2_r, F14_Split2_c:Bool;

act  Customer_arrived,
     Get_customer_identity,
     ID_collected,
     Get_customer_demand,
     Data_collected,
     Store_ID_Data,
     ID_data_stored,
     Store_customerDemand_data,
     Data_stored,
     Credit_worthiness,
     CreditWorthyness_computed,
     Rating_customer,
     Rating_computed,
     Customer_acceptance,
     Customer_accepted,
     Customer_not_accepted,
     Create_optimized_product,
     Price_computed,
     Create_contract,
     Contract_created,
     RM_signature,
     RM_has_signed,
     Customer_signature,
     Contract_signed,
     Approve_contract,
     Contract_approved,
     Cash_payment,
     Cash_paid,
     Cash_returned,
     Open_PP,
     Closed_PP,
     Close_contract,
     Contract_closed;

act  send:DataString#Store#Store;   % Copy information from one storage place to another.
     generate:DataString#Store;     % Generate some information and store it.
     drop:DataString#Store;         % Remove information from some storage place.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                %
%   Parallel combination of all tasks and events                 %
%                                                                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

init allow({send, generate, drop,
            Customer_arrived, Get_customer_identity,
            ID_collected, Get_customer_demand,
            Data_collected, Store_ID_Data,
            ID_data_stored, Store_customerDemand_data,
```

```
            Data_stored, Credit_worthiness,
            CreditWorthyness_computed, Rating_customer,
            Rating_computed, Customer_acceptance,
            Customer_accepted, Customer_not_accepted,
            Create_optimized_product, Price_computed,
            Create_contract, Contract_created,
            RM_signature, RM_has_signed,
            Customer_signature, Contract_signed,
            Approve_contract, Contract_approved,
            Cash_payment, Cash_paid,
            Cash_returned, Open_PP,
            Closed_PP, Close_contract,
            Contract_closed,
            E1_F1_c, F1_E2_c, E2_F2_c, F2_E3_c,
            E3_F3_c, F3_E4_c, E4_F4_c, F4_E5_c,
            E5_F5_c, F5_E6_c, E6_F6_c, F6_E7_c,
            E7_F7_c, Split1_E8_c, Split1_E9_c, E8_F8_c,
            F8_E10_c, E10_F9_c, F9_E11_c, E11_F10_c,
            F10_E12_c, E12_F11_c, F11_E13_c, E13_F12_c,
            F12_E14_c, E14_F13_c, F13_E15_c, E15_F14_c,
            E16_F14_c, Split2_E16_c, Split2_E17_c, E17_F15_c,
            F15_E18_c, F7_Split1_c, F14_Split2_c
           },

         comm({E1_F1_s|E1_F1_r->E1_F1_c,
               F1_E2_s|F1_E2_r->F1_E2_c,
               E2_F2_s|E2_F2_r->E2_F2_c,
               F2_E3_s|F2_E3_r->F2_E3_c,
               E3_F3_s|E3_F3_r->E3_F3_c,
               F3_E4_s|F3_E4_r->F3_E4_c,
               E4_F4_s|E4_F4_r->E4_F4_c,
               F4_E5_s|F4_E5_r->F4_E5_c,
               E5_F5_s|E5_F5_r->E5_F5_c,
               F5_E6_s|F5_E6_r->F5_E6_c,
               E6_F6_s|E6_F6_r->E6_F6_c,
               F6_E7_s|F6_E7_r->F6_E7_c,
               E7_F7_s|E7_F7_r->E7_F7_c,
               Split1_E8_s|Split1_E8_r->Split1_E8_c,
               Split1_E9_s|Split1_E9_r->Split1_E9_c,
               E8_F8_s|E8_F8_r->E8_F8_c,
               F8_E10_s|F8_E10_r->F8_E10_c,
               E10_F9_s|E10_F9_r->E10_F9_c,
               F9_E11_s|F9_E11_r->F9_E11_c,
               E11_F10_s|E11_F10_r->E11_F10_c,
               F10_E12_s|F10_E12_r->F10_E12_c,
               E12_F11_s|E12_F11_r->E12_F11_c,
               F11_E13_s|F11_E13_r->F11_E13_c,
               E13_F12_s|E13_F12_r->E13_F12_c,
               F12_E14_s|F12_E14_r->F12_E14_c,
               E14_F13_s|E14_F13_r->E14_F13_c,
               F13_E15_s|F13_E15_r->F13_E15_c,
               E15_F14_s|E15_F14_r->E15_F14_c,
               E16_F14_s|E16_F14_r->E16_F14_c,
               Split2_E16_s|Split2_E16_r->Split2_E16_c,
               Split2_E17_s|Split2_E17_r->Split2_E17_c,
               E17_F15_s|E17_F15_r->E17_F15_c,
               F15_E18_s|F15_E18_r->F15_E18_c,
               F7_Split1_s|F7_Split1_r->F7_Split1_c,
               F14_Split2_s|F14_Split2_r->F14_Split2_c
              },
           E1||F1||E2||F2||E3||F3||E4||F4||E5
           F5||E6||F6||E7||F7||Split1||E8||E9||
           F8||E10||F9||E11||F10||E12||F11||E13||
           F12||E14||F13||E15||F14||Split2||E15||E17||F15||E18));
```

9 Related Work

In [19] the importance of a resource and data driven analysis of business processes is stressed. But the authors do not deliver a formal solution suitable to be fully automated as requested by Credit Suisse (CS). In [20] disaster recovery plans are evaluated based on ARIS methodology.

This solution does not show how to generate the full configuration space that fulfills possible side-constraints as requested by CS. In [21] an organizational solution to address information security management problems is presented. But this solution cannot be automated as requested by CS. In [22] and [23] the claim is made that continuity processes need to be checked for security, risk and compliance, and that BCMs and risk solutions should be soundly integrated. This claim is fully compatible with the view of CS. In [24] a solution using EPC to simulate processes regarding their risks and costs is proposed by the help of a goal-risk framework. But the complete process configuration space can not be generated and checked for side-constraints as requested by CS.

In [25] the workflow system AgentWork is able to support dynamic workflows based on event-condition-action rules. In contrast to that, CS requested that workflow adaptions should be handled based on declarative continuity snippets only. Given such snippets, we can apply our modification technique automatically. Therefore, such rules do not need to be specified. In [26] a solution guaranteeing the structural correctness of a process model is presented while applying dynamic changes. However, this case is different from the CS scenario where a set of continuity processes is generated in advance based on continuity snippets to enable optimizations and case based decisions, assumed that given side-constraints are respected. In [27] change patterns are proposed as a means to handle modifications of a workflow model and in [28] important correctness problems regarding general modifications are discussed in a comparative survey. In the present scenario already well-formed sub-processes are given. The presented generation technique composes these sub-processes in a controlled way, such that the well-formednes is preserved, which represents an important correctness issue. In addition to that, CS requested to check side-constraints. We do that by the help of graph constraint checks. Therefore, modification rules need not to be maintained and side-constraints can be modeled globally. In [3] a framework for service, process and rule models in the context of enterprise engineering is presented. The techniques presented in this paper are kept fully compatible with this approach as requested by CS.

In [29] and [30] the use of graph transformation and graph substitution techniques is discussed. However, our focus is different. The reconstructed graph grammar formalizes the operational semantics. So, there is no need to model dependencies. They can be automatically derived from the descriptive EPC model. Therefore, the overall modeling effort can be minimized as requested by CS.

10 Conclusions and Future Work

BCMSs have to support the execution of alternatives for regular business processes in case of failures. For this purpose, these alternatives have to be modeled and maintained. However, the modeling of complete alternatives for all combinations of failures is not practicable and inconsistencies may easily occur. Furthermore, security, risk and compliance shall also be ensured for all these alternatives.

The presented solution dramatically reduces the necessary efforts and supports an automatic validation of the objectives in an intuitive and formal way. Alternatives are generated automatically based on a set of declarative fragments that replace regular process parts for particular failures. Complete alternatives for combinations of failures can therefore be derived using the same set of fragments. The business functions of the derived process models are ensured to get correct and available in- and output data and furthermore, the organizational entities are ensured to be able to retrieve the data, because they are required to have access to them. Finally, the graph model specifies which actor executes which business function and which data occurs on which storage devices. This enables automatic checks of security requirements using graph constraints as well as simulations that can be evaluated usind the annotated costs.

Therefore, the presented technique is practicable, easy to maintain and supports a formal validation of the results. Future work will encompass the implementation of the presented graph techniques for process optimization and composition. It will further address more cases as well as their validation.

References

[1] Knight, Pretty: The impact of catastrophes on shareholder value. In: The oxford executive research briefings, University of Oxford, Oxford, England, Templeton College (1996)

[2] Chandramouli, R.: Enterprise access policy enforcement for applications through hybrid models and xslt technologies. In: ICEC '04: Proc. 6th Int. Conf. on Electronic commerce, New York, NY, USA, ACM (2004) 490–499

[3] Brandt, C., Engel, T., Hermann, F.: Security and consistency of it and business models at credit suisse realized by graph constraints, transformation and integration using algebraic graph theory. In: BPMDS 2009 and EMMSAD 2009, LNBIP 29, Berlin/Heidelberg, Springer (2009) 339–352

[4] Sarbanes, P., Oxley, M.: Public company accounting reform and investor protection act, Washington, Government Printing Office (2002)

[5] BSi: Business continuity management. bsi 25999-1, British Standards Institution (2006)

[6] Boehmer, W.: Survivability and business continuity management system according to bs 25999. In: Proc. Int. Conf. on Emerging Security Information, Systems and Technologies (SECURWARE 2009), Athens/Vouliagmeni, Greece, IEEE Computer Society (June 2009)

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

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

[9] Biermann, E., Ermel, C., Lambers, L., Prange, U., Taentzer, G.: Introduction to agg and emf tiger by modeling a conference scheduling system. Software Tools for Technology Transfer (2010) To appear.

[10] Tiger Project Team, Technische Universität Berlin: EMF Tiger (2009) http://tfs.cs.tu-berlin.de/emftrans.

[11] Corradini, A., Hermann, F., Sobociński, P.: Subobject Transformation Systems. Applied Categorical Structures **16**(3) (February 2008) 389–419

[12] Hermann, F.: Permutation Equivalence of DPO Derivations with Negative Application Conditions based on Subobject Transformation Systems. In: Proc. Int. Conf. on Graph Transformation-Doctorial Symposium (ICGT-DS'08), Electronic Communications of the EASST (2009) (to appear).

[13] Hermann, F., Corradini, A., Ehrig, H., König, B.: Efficient Analysis of Permutation Equivalence of Graph Derivations Based on Petri Nets. In Küster, J., Tuosto, E., eds.: Proc. Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT'10), EC-EASST (2010) To appear.

[14] Brandt, C., Engel, T., Hermann, F.: Business Continuity Management regarding Security, Risk and Compliance of a Loan Granting Process at Credit Suisse using Algebraic Graph Theory (Long Version). Technical report, Technische Universität Berlin,Fakultät IV ((to appear 2009)) tfs.cs.tu-berlin.de/publikationen/Papers09/BEH09.pdf.

[15] Groote, J.F., Mathijssen, A., Reniers, M.A., Usenko, Y.S., van Weerdenburg, M.: 2. In: Analysis of Distributed Systems with mCRL2. 1.st. edn. Chapman & Hall/CRC (2008) 99–128 (`www.mcrl2.org`)

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

[17] Ehrig, H., Ermel, C., Hermann, F.: On the Relationship of Model Transformations Based on Triple and Plain Graph Grammars. In Karsai, G., Taentzer, G., eds.: Proc. Third International Workshop on Graph and Model Transformation (GraMoT'08), New York, NY, USA, ACM (2008)

[18] Groote, J., Willemse, T.: Model-checking processes with data. Science of Computer Programming **56** (2005) 251–273

[19] Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. IBM Systems Journal **42**(3) (2003)

[20] Sztandera, P., Ludzia, M., Zalewski, M.: Modeling and analyzing disaster recovery plans as business processes. In: Computer Safety, Reliability, and Security, Lecture Notes in Computer Science. Volume 5219., Berlin/Heidelberg, Springer (2008) 113–125

[21] Eloff, J.H.P., Eloff, M.: Information security management: a new paradigm. In: Proc. research conf. of the South African Institute of Computer Scientists and Information Technologists on enablement through technology (SAICSIT '03), Republic of South Africa, South African Institute for Computer Scientists and Information Technologists (2003) 130–136

[22] Quirchmayr, G.: Survivability and business continuity management. In: Proc. of the 2nd WS on Australasian information security, Data Mining and Web Intelligence, and Software Internationalisation (ACSW Frontiers '04), Darlinghurst, Australia, Australia, Australian Computer Society, Inc. (2004) 3–6

[23] Cha, S.C., Juo, P.W., Liu, L.T., Chen, W.N.: Riskpatrol: A risk management system considering the integration risk management with business continuity processes. In: Intelligence and Security Informatics, Taipei, IEEE (2008) 110 – 115

[24] Asnar, Y., Giorgini, P.: Analyzing business continuity through a multi-layers model. In: Business Process Management, Lecture Notes in Computer Science. Volume 5240., Berlin/Heidelberg, Springer (2008) 212–227

[25] Müller, R., Greiner, U., Rahm, E.: AGENT WORK: a workflow system supporting rule-based workflow adaptation. Data Knowl. Eng. **51**(2) (2004) 223–256

[26] Reichert, M., Dadam, P.: ADEPT flex -supporting dynamic changes of workflows without losing control. Journal of Intelligent Information Systems **10**(2) (1998) 93–129

[27] Weber, B., Reichert, M., Rinderle-Ma, S.: Change patterns and change support features - enhancing flexibility in process-aware information systems. Data Knowl. Eng. **66**(3) (2008) 438–466

[28] Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems: a survey. Data Knowl. Eng. **50**(1) (2004) 9–34

[29] Heimann, P., Joeris, G., Krapp, C.A., Westfechtel, B.: DYNAMITE: dynamic task nets for software process management. In: ICSE '96: Proceedings of the 18th international conference on Software engineering, Washington, DC, USA, IEEE Computer Society (1996) 331–341

[30] Bogia, D.P., Kaplan, S.M.: Flexibility and control for dynamic workflows in the worlds environment. In: COCS '95: Proceedings of conference on Organizational computing systems, New York, NY, USA, ACM (1995) 148–159