# Evaluation of a Business Continuity Plan using Process Algebra and Modal Logic

*Wolfgang BOEHMER, Christoph BRANDT and Jan Friso GROOTE*

*Department of Computer Science, Security Engineering Group, Technische Universität Darmstadt, Germany and*

*Computer Science and Communications Research Unit, University of Luxembourg and*

*Department of Mathematics and Computer Science, Eindhoven University of Technology*

September 22, 2009

**Abstract**

Since (1996) Knight and Pretty published their study about the impact of catastrophes on shareholder value, the need for a business continuity management system (BCMS) became clear. Once a BCMS is in place, the corresponding risks can be insured against. The BS 25999 certificate can serve as proof of implementation. It requires defined business continuity plans (BCP). However, processes based on BCPs are rarely tested. Therefore, little knowledge is available to confirm their proper functioning and their non-functional properties.

This paper addresses the verification of BCPs. We show how to model, simulate and verify normal business processes and business processes that are based on a BCP. As a formal method, we use process algebra and modal logic to explain the semantics of conceptual business process models. Our study places emphasis on questions regarding the potential capacity and duration of a process based on a BCP as well as those of an organizational security policy. By doing this, we are able to demonstrate that ex-ante evaluation is not only possible but also effective.

## 1 Introduction

The problem statement is about interruptions of the central value chain (CVC) a company is running as well as their handling. Such interruptions can be caused by catastrophies or other major and rare incidents. We assume, that a company is only able to survive a certain amount of time without its CVC. A business continuity management system (BCMS) requires that a business continuity plan (BCP) is put in place. There are different BCPs for different kinds of incidents that caused an interruption. The underlying idea is that a BCP keeps the company alive while it is recovering by the help of a disaster recovery plan (DRP). Because incidents do not occur often, there is little proven knowledge about the proper functioning of a BCP. Only some rare, non-representive evidence out of real BCP-excercises is available. In order to understand that a BCP will operate well in case of a disaster, a priori analysis techniques are required. Today's best practices do not provide solutions for this.

In this paper we want to experiment with formal techniques to describe a BCP and to analyse various aspects of it. More concretely we look at the process algebraic specification language mCRL2 [10] to describe the behaviour and use modal logic and visualisation techniques [19] to get insight into it. For this purpose we introduce a small exemplary business process and a business continuity plan, formalize and analyze them. In particular, the business continuity plan has various options which we want to compare.

The first question, we like to discuss, is whether the BCP is able to backup a predefined amount of business cases in a certain time span compared to the amount the CVC usually handles. We also want to know whether early filtering of not so valuable customers will improve throughput.

The second question is about certain organizational security policies. So, we ask ourselves a typical security property, namely in which cases the four-eyes principle holds. The four-eyes principle says that essential business decisions must be approved by two separate individuals in an organisation. The second property is a lifeness property. We want to know whether all customers are served properly and completely under all circumstances.

The third question came upon us, when looking at the visual representation of the behaviour of the systems. We observed very particular behavioural patterns. Inspired by this we want to know which decisions in the business process cause these behavioural patterns. One of these questions, formulated in terms of the business process, relates to the quite different end-times at which customers will have been served. We want to know which activities in the business process are responsible for these end-times.

At a higher level, the contribution of this paper consists of a new semantical interpretation of the formal semantics of a CVC and a BCP in advanced process algebra, a simulation of their behavior to determine their capacity, the application of new visualisation techniques and a proof that fully automated model checking using modal logic of safety and liveness properties can readily be done.

The paper is organized as follows: First, we introduce the context of a Business Continuity Management System (BCMS) that is handling the BCP and DRP of the central value chain of a company. Second, we present a loan granting process (LGP) as an example of a critical business process. Third, we demonstrate how to use mCRL2, an advanced process algebra with a sophisticated modal logic for model checking. Fourth, we discuss how to define the formal semantics of the LGP and show state space exploration and simulation results regarding the performance capacities of the process. We further present model checking results to show compliance with an organizational security policy (segregation of duty) that can be proven automatically. Sixth, we reference some related work and summarize our conclusions as well as prospective future work.

## 2 Business Continuity Managment System (BCMS)

A business continuity management is a holistic management process that identifies potential impact that threaten an organization and provides a framework for building resilience and capability for an effective response that safeguards the interests of its key stakeholders, reputation, brand and value creating activities [27]. The fundamental idea of a BCMS is based on the fact that the Business Continuity Management (BCM) is meant to manage kinds of rare business risks with a huge impact on a company. The BCMS is capable of responding adequately in extreme situations (catastrophic events) with pre-defined plans (BCP).

So, a company that wants to safeguard their critical value chain, should focus on securing revenues by taking adequate risk countermeasures. Since 2007, the BS 25999-2:2007 [28], published by the British Standard Institution (BSI), is available. It is an industry-wide recognized best-practice method that governs the creation of a BCMS. It encompasses a BCP and a DRP (Disaster Recovery Plan). In addition to that, the continuity processes should be compliant with organizational security policies (e.g. segregation of duty). We will show how these qualities can be analyzed ex-ante by the help of formal methods.

### 2.1 Concrete Example

A simple real-world example is used here to illustrate the concept of a management-system. A person who wants to manage his or her weight by the help of a management-system focusses on the consumed and burned calories. A possible objective can be to balance these values. Figure 1 illustrates the idea of a balanced system.

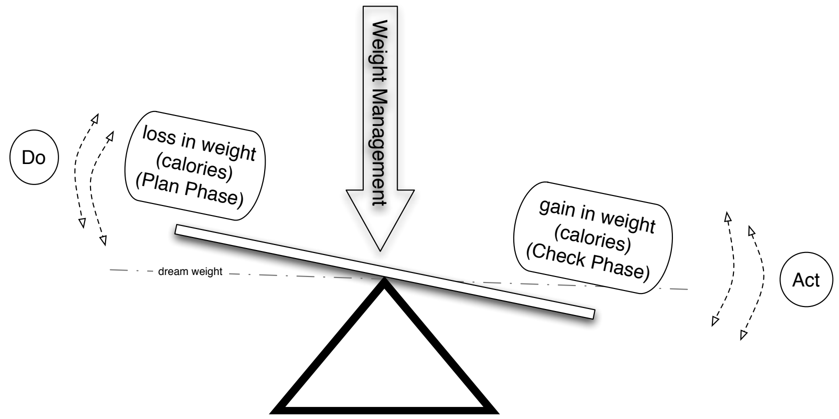
Weight Management
loss in weight (calories) (Plan Phase)
gain in weight (calories) (Check Phase)
Do
Act
dream weight

Figure 1: Weight management system and ups and down as a seesaw

Another objective can be to reduce the weight of a person. In this case there must be more calories burned than consumed. The measuring instrument is the weighing machine. In the next section we will argue how this approach can be mapped to the concept of a Business Continuity Management System.

## 2.2 Basic idea of a BCMS according BS25999

The standard requires the implementation of a management system in accordance with the PDCA cycle (Plan-Do-Check-Act), as well as those already required in the standards ISO 27001 and ISO 20000 and other standards. The aim of the PDCA cycle is the idea of imperfection, and follows a continuous improvement process. In the check-phase it is examined whether the plan is still in line with the objectives. If not, the corrections are resolved in the Act-period.

Figure 2 sketches the operational view on a PDCA cycle regarding an underlying BCMS. A BCMS is a framework that helps to keep the balance between the risks (potential disasters and impacts on the critical business process) and the available countermeasures (business continuity processes and business recovery processes) respecting the MTPD as a real-world side constraint.

The timespan available for the BCP to deliver a minimum business revenue is defined as the maximum tolerable period of disruption (MTPD). The continuation of the value chain at an acceptable level for a defined period ($\Delta t$) is ensured.

BCM is a reactive process that becomes active only after the occurrence of the disaster. In this context, the maximum allowable down time MTPD, which starts running after the occurrence of the disaster, increases considerably in importance. The MTPD is determined using the length of time the critical activities of the value chain require to being working again after the occurrence of a disaster, so the company can survive (see figure 3). This period of time ($\Delta T_{max} = t_3 - t_0$) is an ultimate boundary for a company and decides the company's survival. If this ultimate limit is exceeded, the company is irretrievably lost (see curve (2), figure 3). The relation between critical activities and the value chain is determined by the Business Impact Analysis (BIA). Within the BIA, the dependent critical resources (key stake holders, key products, key services) and their importance to the critical activities (core processes of the value chain) are analyzed. A BCMS includes the vital business processes. Recovering only the working infrastructure, e.g. replacing a failed IT infrastructure by an emergency one, will not meet a BCMS, as the IBM report clearly pointed out [12]. We will discuss the business process capacity (BPC) of the CVC and different variants of continuity processes.

Figure 3 shows a qualitative representation on a money/timeline what happens if a disaster strikes

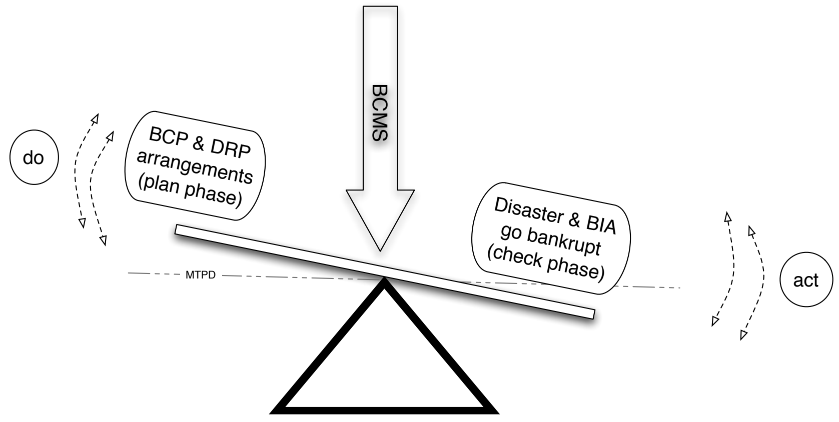
BCMS
do
BCP & DRP arrangements (plan phase)
Disaster & BIA go bankrupt (check phase)
MTPD
act

Figure 2: Business Continuity Management System (BCMS) and ups and down as a seesaw

at time $t_0$. This shows that immediately after the occurrence of a disaster the calculated turnover collapses. At time $t_1$ the processes of the BCP (emergency operation) start and create a turnover at an acceptable level. A little later, at time $t_2$ the recovery processes start and at time $t_4$ the company is back to its normal levels of operations. The dash dotted line in the figure shows that the costs after a disaster increase. In the event that no countermeasures (BCP, DRP) are taken, or that the countermeasures do not work, the costs continue to increase (see curve (2)). The ideal situation is that the Business Continuity Plan and the Disaster Recovery Plan work so well, that costs are as depicted in curve (1).

If no action (BCP, DRP) has been taken at the time $t_3$, or has not been started until the time $t_3$, then the costs will increase until company insolvency is reached (cf. figure 3). The costs are determined by the obligations of the company. These consist of personnel, technical expenses and the cost of delivery, performance, or possibly storage costs, etc.

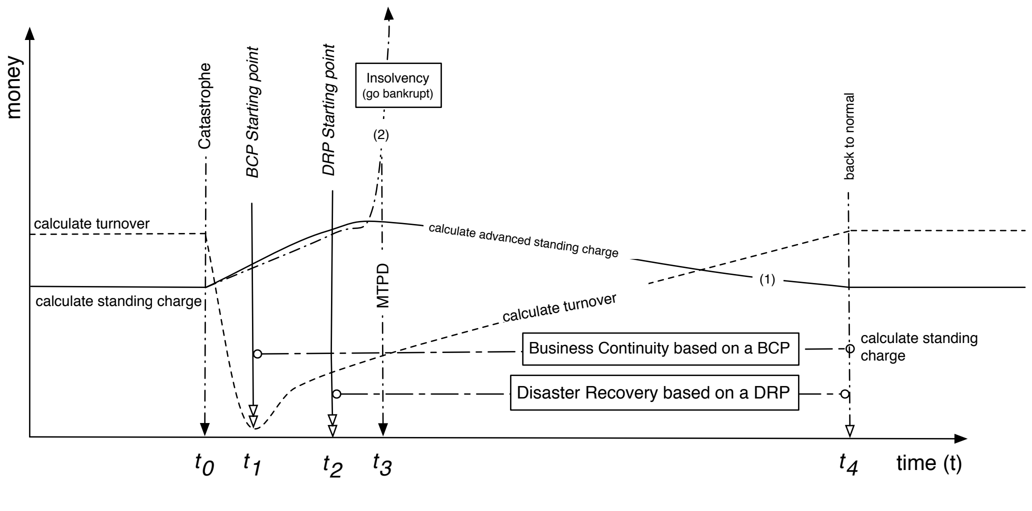

Figure 3: Illustration of aspects of a catastrophe ($t_0$) and the reaction ($t_1, ..., t_4$)

It is in the self interest of a company to keep the BCPs and DRPs operational. Usually, this is tested on a regular basis by simulating that something goes astray within the ordinary business process. Because these tests are expensive, they are not executed very often and generally only address certain aspects of the recovery plans. Therefore, this testing only provides a rather haphazard prediction of the effectiveness of such plans if a true disaster strikes.

But the continuation of business is so important that deeper analyses are required. In the literature, there are generally two different methods available to measure the performance.

- On the one hand, the performance can be measured on the maturity of processes, such as Spice (ISO/IEC 15504) or CMMI.
- On the other hand, the performance an be measured on the basis of appropriate indicators (KPI). Proposals for the handling of key indicators are found in the literature [2, 23].

It is however difficult to come to a reliable assessment of such indicators. Assessments based on simulated business failures and tests with BCPs and DRPs do not take place sufficiently often to obtain reliable figures. The inevitable alternative is the application of good judgement and educated guessing.

In order to improve the quality of the analysis of BCPs and DRPs one should model these and the ordinary business process such that they can be simulated. The first ideas of how to do this have been presented in [5]. If the business continuity processes are in place and capacity and throughput volumes are known, this can be modeled using process algebra and abstract data. With modal logic safety and liveness properties can then be reviewed. These ideas will be pursued in the next sections.

# 3 Loan Granting Business Process

We like to discuss a simplified version of a loan granting process (LGP) that can be found in an arbitrary bank. First, we present the normal business situation. Second, we discuss the business continuity situation.

## 3.1 The Normal Business Situation

In a normal business situation (NBS) a LGP consists of a couple of procedures: First, a processing clerk (PC) identifies the customer (C) in front of him. We assume, that this activity will take about 30 min (T=30). In our formal model we will make reference to this activity by F1 (A=F1). Second, the PC accepts the supporting documents from C (A=F2, T=10). According to our process model (see figure 4 and figure 5), these activities can run sequentially or simultaneously. Third, the PC stores the data about C into the database DB1 (A=F3a, T=5). Fourth, the creditworthiness of C is checked by the PC making a request to an external credit office (A=F3, T=60). As a result of this request C can be accepted (E2a) or rejected (E2b). Fifth, the post processing clerk (PPC) determines the rating for C (A=F4, T=15) by using data from DB1 and support from a rating application. Sixth, the PPC and the supervisor (S) create an optimized contract (A=F5, T=20). To do this, they access the price engine application and the output management application (OMA). Seventh, the manager (M) prints the contract (A=F6, T=5). The result is an unsigned contract. Finally, C and M both sign the contract (A=F7, T=30).

## 3.2 The Business Continuity Situation

In a business continuity situation (BCS) a LGP consists of a couple of slightly different procedures. We assume here, that the business continuity process will take care of the fact that the business critical IT systems are not available. The overall purpose it to make a minimum business functionality available.

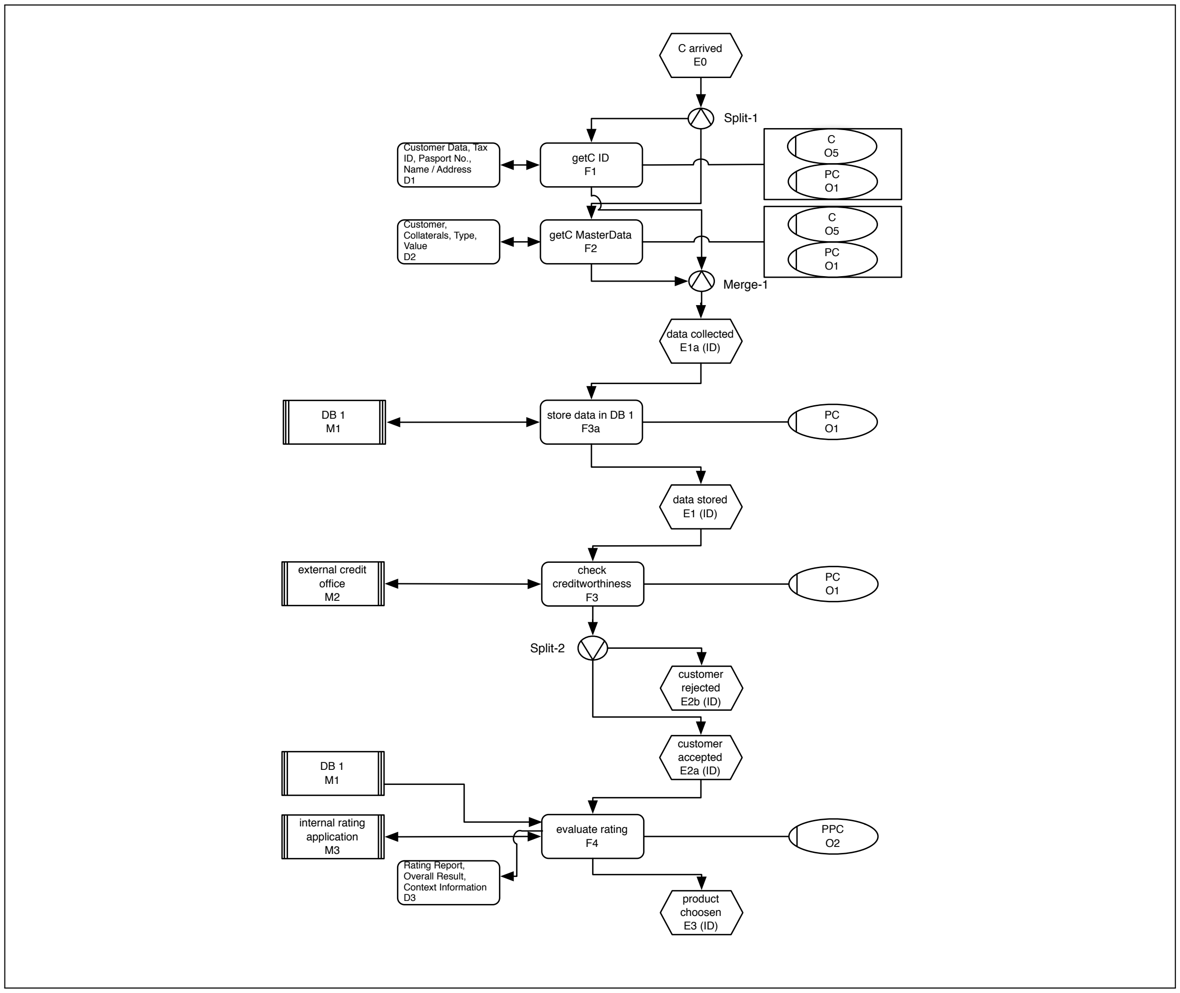

Figure 4: Normal Business Process – Part 1

| Process | Sending to ... | Receiving from ... | Time |
|---|---|---|---|
| E0 | Split1 | C, ETC | 0 |
| Split1 | F1, F2 | E0 | 0 |
| F1 | Merge1, PC, C | Split1, PC, C | 30 |
| F2 | Merge1, PC, C | Split1, PC, C | 10 |
| Merge1 | E1a | F1, F2 | 0 |
| E1a | F3a | Merge1 | 0 |
| F3a | E1, PC | E1a, PC | 5 |
| E1 | F3 | F3a | 0 |
| F3 | Split2, PC | E1, PC | 60 |
| Split2 | E2a, E2b | F3 | 0 |
| E2a | F4 | Split2 | 0 |
| F4 | E3, PPC | E2, PPC | 15 |
| E3 | F5 | F4 | 0 |
| F5 | E4, S, PPC | E3, S, PPC | 20 |
| E4 | F6 | F5 | 0 |
| F6 | E5, M | E4, M | 5 |
| E5 | F7 | F6 | 0 |
| F7 | E6, M, C | E5, M, C | 30 |
| E6 | C, ETC | F7 | 0 |

Table 1: Partial Process Algebra View on the Normal Process

### 3.2.1 Variant A.1 and A.2

The first two variants of the business continuity process we like to present are defined as follows:

A PC identifies C (A=F1, T=30) first. Second, PC collects the supporting documents from C (A=F2, T=10). These two activities can be done sequentially or in parallel. Third, PC makes a decision if the customer is accepted or rejected (A=F4a, T=60). If C is rejected he drops out of the process. Fourth, either PC creates a standard contract for C (variant A.1) or M does (variant A.2; A=F5a, T=90). Finally, C and M sign the contract (A=F7, T=30).

### 3.2.2 Variant B

The third variant of the business continuity process contains an additional first step. During this step the PC determines the business value of C to make the decision if the customer is picked for the business continuity process or returned. The rationale behind this decision is to process only customers with a high business value for the bank to meet the minimum business objectives in the continuity case.

In detail, PC checks the business value of C (A=F0, T=60). Afterwards the process continues as already described in variant A.1 by identifying the customer (A=F1, T=30) and collecting the supporting documents (A=F2, T=10).

## 4 Process Algebra: mCRL2

We like to introduce mCRL2 [8] as a suitable formal platform to define the semantics of event driven process chains [24] used to model critical business processes.

mCRL2 is a process algebra encompassing data and time, provided with a modal logic. It is therefore helpful to specify and analyse a broad range of sytems. As presented in [8], the process algebraic

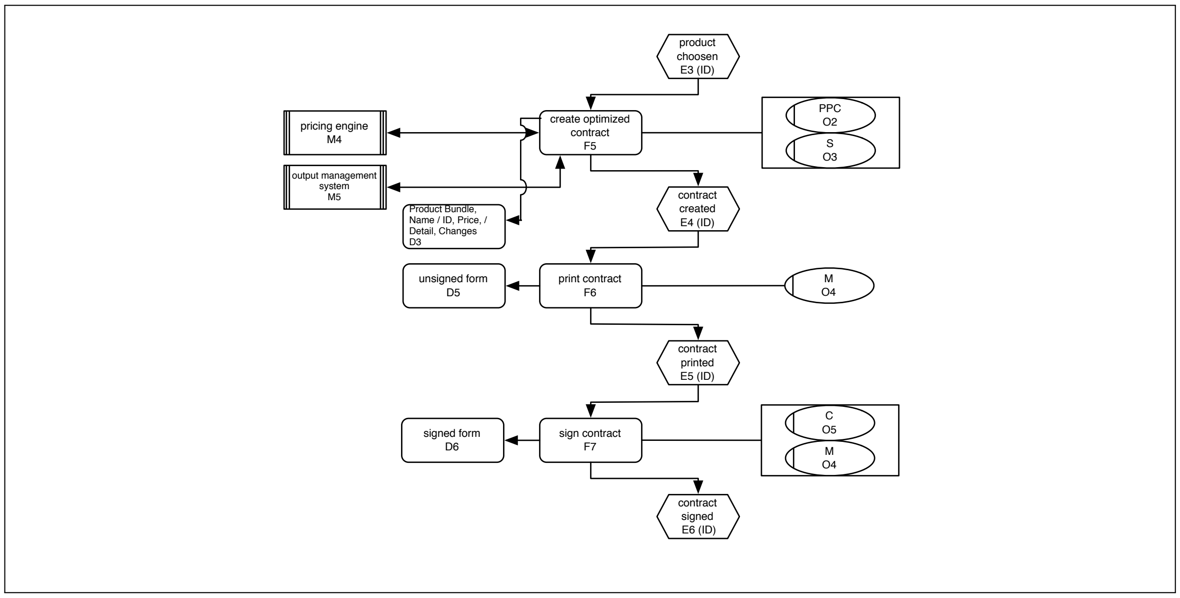

Figure 5: Normal Business Process – Part 2

structure helps to formally specify the communication between subcomponents of a system without touching the rest of it. Therefore, component-based and hierarchical systems are well supported.

## 4.1 The Process Language

The primary notion in the mCRL2 process language [8] is an *action*, which represents an elementary activity or a communication of some systems.

Multiactions enable the specification of actions that are executed together. As a consequence, these multiactions allow the separation of parallelism and communication. When two actions can execute at the same time, a multiaction with those actions is the result. Communication can then be applied to these multiactions to make certain actions communicate with each other.

## 4.2 The Data Language

The mCRL2 data language [8] is a functional language based on higher-order abstract data types, extended with concrete data types: standard data types and sorts constructed from a number of type formers. It can be used to parameterize processes and actions. Because the data-language is higher order, functions are first-class citizens and can therefore be used just as easily as other data [10, 11].

## 4.3 The Modal $\mu$-Calculus

By adding explicit minimal and maximal fixed point operators to Hennessy-Milner logic, the modal $\mu$-calculus [11] is obtained. Modal formulas are extended with data, similar to processes. So, modal variables can have arguments, actions can carry data arguments as well as time stamps and existential and universal quantification is possible.

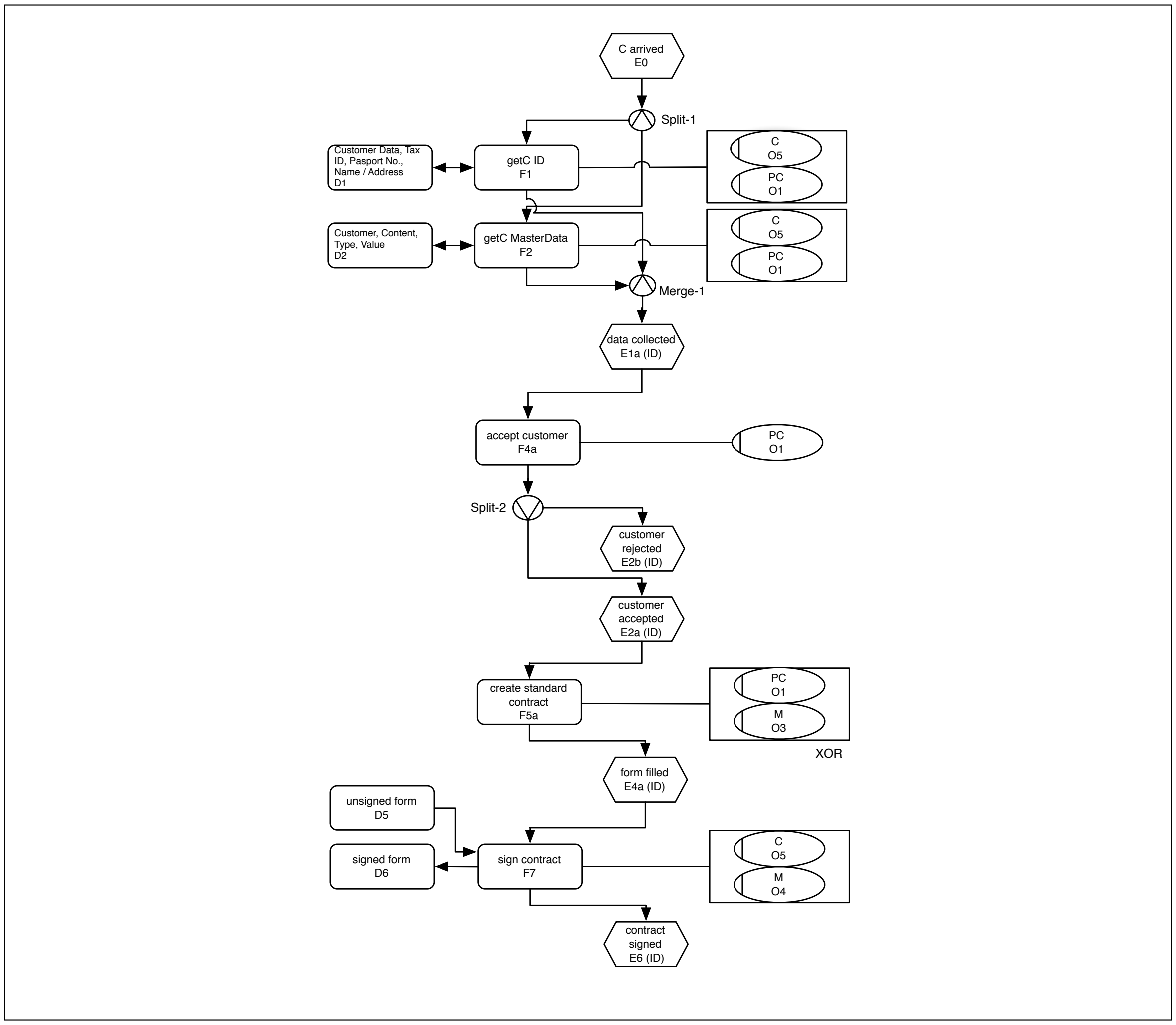

Figure 6: Business Continuity Process – Variant A.1 and A.2

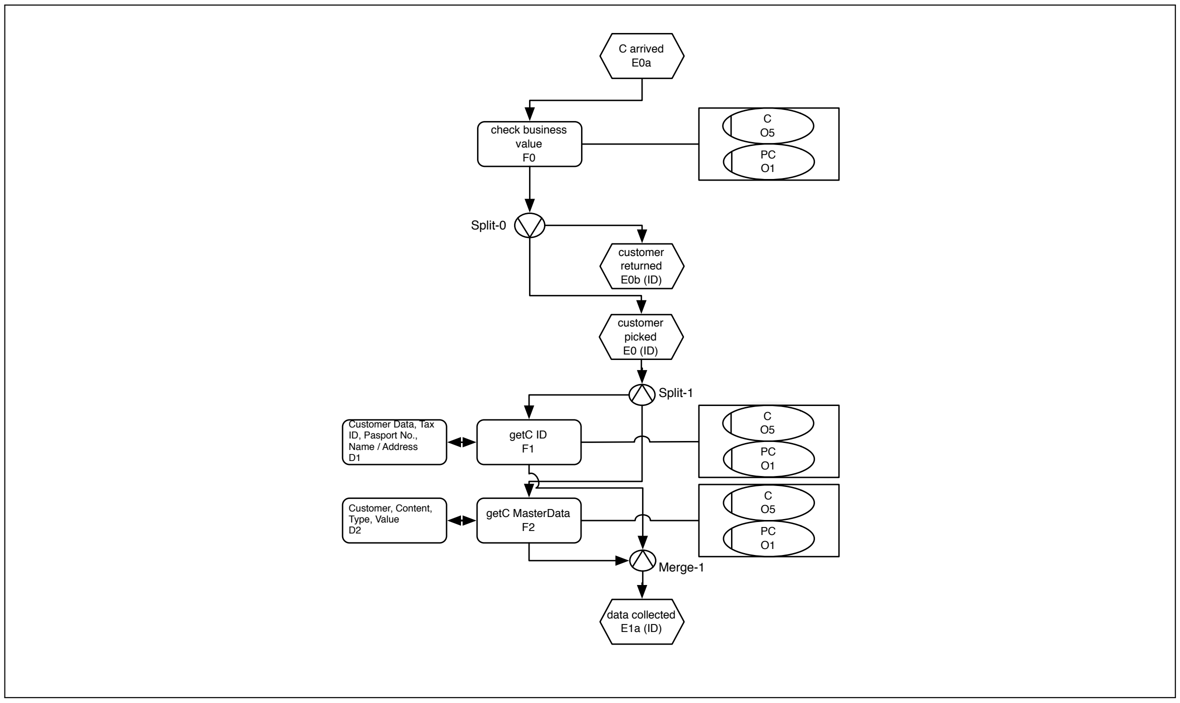

Figure 7: Business Continuity Process – Variant B, Part 1

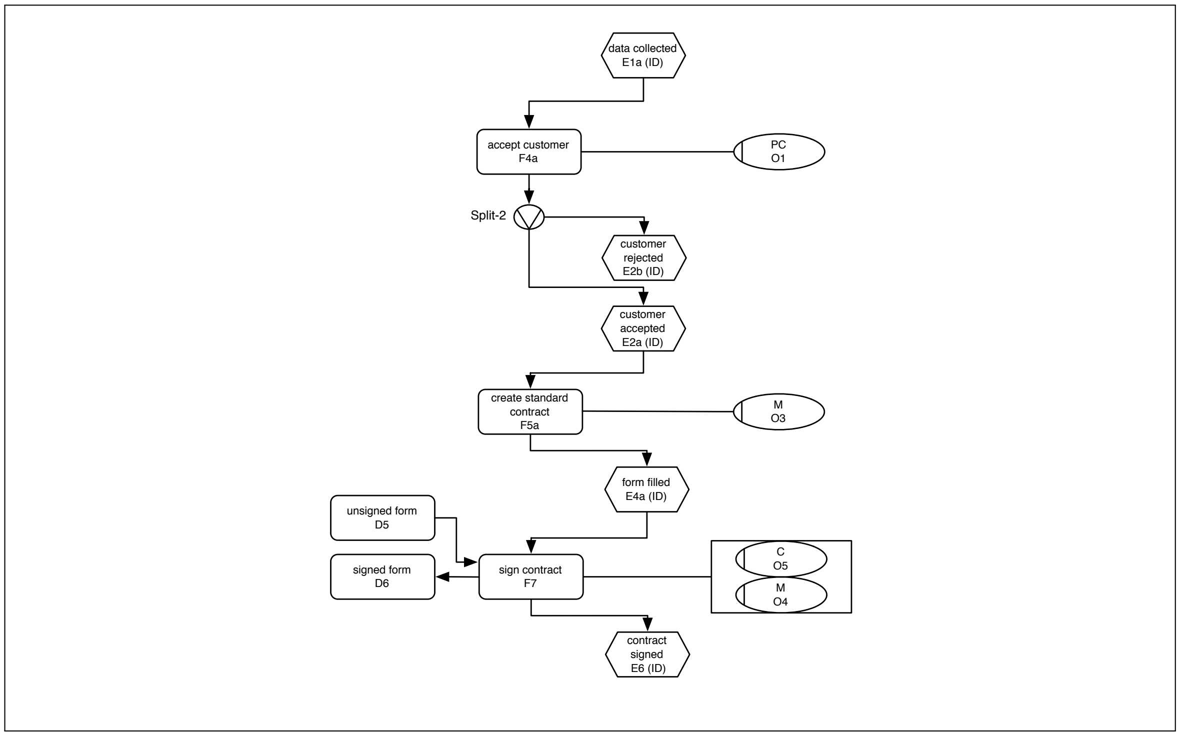

Figure 8: Business Continuity Process – Variant B, Part 2

| Process | Sending to ... | Receiving from ... | Time |
|---|---|---|---|
| E0 | Split1 | C, ETC | 0 |
| Split1 | F1, F2 | E0 | 0 |
| F1 | Merge1, PC, C | Split1, PC, C | 30 |
| F2 | Merge1, PC, C | Split1, PC, C | 10 |
| Merge1 | E1a | F1, F2 | 0 |
| E1a | F4a | Merge1 | 0 |
| F4a | Split2, PC | E1a, PC | 60 |
| Split2 | E2a, E2b | F4a | 0 |
| E2a | F5a | Split2 | 0 |
| E2b | C, ETC | Split2 | 0 |
| F5a_alt1 | E4a, PC | E2a, PC | 90 |
| F5a_alt2 | E4a, M | E2a, M | 90 |
| E4a | F7 | F5a | 0 |
| F7 | E6, M, C | E4a, M, C | 30 |
| E6 | C, ETC | F7 | 0 |

Table 2: Partial Process Algebra View on the Emergency Process

## 4.4 Example: Dining Philosophers

The use of mCRL2 can easily be demonstrated by modeling the dining philosopher problem (DPP) [6]. In the following case the DPP consists of three philosophers and three forks that are shared. Each philosopher has a plate. Forks are placed on the left and right side of every plate. The dishes that are served require two forks to be eaten.

In the following model the sort *PhilId* contains the philosophers. The $n$th philosopher is denoted by $p_n$. The sort *ForkId* containts the forks ($f_n$ denotes the $n$th fork). The functions *lf* and *rf* designate the respective left and right forks of each philosopher.

The process *Phil($p_n$)* models the behavior of the $n$th philosopher. It first takes the left and right forks (in any order; possibly at the same time), then eats, subsequently puts both forks back (again in any order) and finally repeats its own behavior.

The process *Fork*($f_n$) defines the behavior of the $n$th fork. It can perform *up*($p, f_n$) for any philosopher, meaning that the fork is being picked by philosopher $p$. Then it performs *down*($p, f_n$), meaning that the same philosopher puts the fork down and repeats its own behavior.

The system consists of three *Phil* and *Fork* processes that run in parallel. Communication between *get* and *up* as well as between *put* and *down* is enforced. The resulting communication is *look* and *free*. The communication operator, $\Gamma_C(p)$, allows communication only when possible. The blocking operator, $\partial_B(p)$, ensures that nothing else happens.

The mCRL2 toolset is used to generate the state space to be able to check certain properties of the system. Additionally, modal formulas in the $\mu$-calculus can express desired (temporal) properties. These are solved by transforming the system behaviour and formula to a parameterized boolean equation system that is subsequently solved.

By model checking a dead-lock in the dining philosophers can easily found, yielding to trace lock($p_1$,$f_1$) $\cdot$ lock($p_2$,$f_2$) $\cdot$ lock($p_3$,$f_3$). The detected dead-lock represents the situation when each of the philosophers has taken one fork and waits for another one without being able to put the first one back.

One solution is to cross the arms of one of the philosophers The result is a simple LTS consisting of 36 states and 104 transitions without any deadlock states.

```
sort   PhilId = struct p1 | p2 | p3;
       ForkId = struct f1 | f2 | f3;
map    lf, rf : PhilId -> ForkId;
eqn    lf(p1) = f1; lf(p2) = f2; lf(p3) = f3;
       rf(p1) = f3; rf(p2) = f1; rf(p3) = f2;
act    get, put, down, lock, free : PhilId × ForkId;
       eat: PhilId;
proc   Phil(p : PhilId)=
           (get(p,lf(p)) | get(p,rf(p))).eat(p).
           (put(p,lf(p)) | put(p,rf(p))).Phil(p);
       Fork(f : ForkId) =
          sum p : Phil up(p,f).down(p,f).Fork(f);
init   block({get,put,up,down},comm({get|up ->lock,put|down -> free},
           Phil(p1) | Phil(p2) | Phil(p3) |
           Fork(f1) | Fork(f2) | Fork(f3) ));
```

Figure 9: mcrl2 Specification of Dining Philosophers

## 5 Case Study

In this case study (CS) we will investigate the dynamic behavior of the CVC as well as of A.1, A.2 and B. We present the scenario, some snippets of the mCRL2 specification, the process capacity as well as a safety and liveness requirement. The full model and all details are given in the appendix.

### 5.1 The Scenario

Apart from the pure process description, we assume, that there are two process instances running in parallel with two clerks (PC), one post processing clerk (PPC) for back office work, one supervisor (S) for controlling jobs and one manager (M) who can sign contracts. In the business continuity situation there is only one PC that is full-time and one M that is part-time available. We assume that there are no PPC and S available in that case.

### 5.2 The Specification

In the mCRL2 specification of E1 (fig. 10), a customer is passing through the proc instance. It first communicates with F3a, then makes the assumption that some information is stored in a file by c and communicates with F3 before restarting automatically. According to the specification of F1 a customer and a clerk are assigned to this task once the split process has fired. Afterwards, the activity F3 is finished and both are released. The merge process takes over the customer with the time the customer spent in F1. Afterwards F1 restarts.

In the mCRL2 specification of customer (fig. 11), a customer is first created by the event E0, it is then assigned and released by different processes. Finally, it drops out of the business process once it reaches the event E6. PClerks are basically assigned and released by different tasks. Because we assume that there are two distinct clerks we have two data structures here and two separate instantiations.

### 5.3 Capacity Estimations

By simulation we are able to obtain the latest and average customer serving endtimes for the business processes. In table 3 we write it down. Customers arrive every 10 minutes. Some of them are eligible for a loan and others are not (indicated by a †). The endtimes are obtained by highway state space exploration of width 10.000 [7]. The average times are determined using random simulation.

Our interpretation of table 3 is the following: A.2 is safer, and not so bad at all on average, although its potential end serving times look bad, compared to A.1. As an alternative for A.1 we consider B, where the filtering of customers is done initially. We see that this has a mixed effect on the average

```
proc E1=
    sum c:Customer, t:Time.F3a_E1_r(c,t).information_stored_in_file(c,t).
      E1_F3_s(c,t).E1;

map duration_F1:Time;
eqn duration_F1=30;

proc F1=
    sum c:Customer, t:Time.Split_F1_r(c,t).
    sum e:PClerk, u,u':Time.assign_pclerk_r(e,t,u,u').
    sum v,v':Time.assign_customer_r(c,u',v,v').
    % Perform task
    release_customer_s(c,v'+duration_F1).release_pclerk_s(e,v'+duration_F1).
    F1_Merge_s(c,v'+duration_F1).F1;
```

Figure 10: mCRL2 Specification for E1 and F1

```
proc Customer(c:Customer)=
      sum t:Time.E0_3(c,t).Customer(c,t);

  Customer(c:Customer,t:Time)
    =   sum t':Time.assign_customer_s(c,t',t,max(t,t')).
        sum u:Time.release_customer_r(c,u).Customer(c,u)
    +   sum t:Time.E6_r(c,t).delta;

sort PClerk=struct pc1 | pc2;

proc PClerk(e:PClerk,t:Time)=
    sum t':Time.assign_pclerk_s(e,t',t,max(t,t')).
    sum u:Time.release_pclerk_r(e,u).PClerk(e,u);

proc PClerks=PClerk(pc1,0) || PClerk(pc2,0);
```

Figure 11: mCRL2 Specification for Customer and PClerk

| Process | 1† | 2 | 3† | 4 | 5 | 6† | 7 | 8† | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| CVC | 570 | 670 | 595 | 665 | 670 | 595 | 670 | 590 | 665 | 675 |
| BCP A.1 | 1540 | 1570 | 1540 | 1570 | 1570 | 1540 | 1570 | 1540 | 1570 | 1570 |
| BCP A.2 | 1000 | 2420 | 1000 | 2340 | 2300 | 1000 | 2360 | 1000 | 2340 | 2340 |
| BCP B | 60 | 1770 | 1650 | 1770 | 1770 | 1650 | 1770 | 1650 | 1770 | 1770 |
| Process | 1† | 2 | 3† | 4 | 5 | 6† | 7 | 8† | 9 | 10 |
| CVC | 283 | 459 | 434 | 473 | 503 | 378 | 522 | 461 | 465 | 536 |
| BCP A.1 | 729 | 1151 | 893 | 1105 | 1182 | 870 | 981 | 819 | 1165 | 1356 |
| BCP A.2 | 475 | 1018 | 671 | 1049 | 1422 | 756 | 986 | 677 | 1386 | 1464 |
| BCP B | 60 | 1275 | 344 | 1469 | 1427 | 632 | 1347 | 359 | 1548 | 1462 |

Table 3: Latest and average customer serving endtimes

```
forall e:PClerk, c:Customer.
        [true*.F1_getC_ID(e,c).true*.F3_check_credit_worthiness(e,c)]false
```

Figure 12: mCRL2 Specification of the segregation of duty (4-eye-principle)

througput times, but hardly any on the latest times customers can be in the system. Note that B is a variant on A.1. Clearly, this allows us to study alternative process models and enables us to develop an optimal BCP.

## 5.4 A Safety Property

According to Kindler [14] a safety property expresses informally that something (bad) will *not* happen. In our case, we like to prove that the clerk who identifies the customer will never be the one who judges his or her creditworthiness. Therefore, we can conclude that the segregation of duty (4-eye-principle) is respected here. The corresponding modal $\mu$-formula that can be checked by the help of the mCRL2 toolsuite is given in figure 12.

## 5.5 A Liveness Property

In the same way, a liveness property can be informally defined [14]. It expresses that eventually something (good) *can* or *must* happen. In our case, we like to prove that every client will always be served eventually. The corresponding modal $\mu$-formula that can be checked by the help of the mCRL2 toolsuite is given in figure 13.

The safety formula only holds in scenario BCP A.2. In case BCP A.1 and B the 4-eye principle is not respected. The liveness property fortunately holds in all scenarios.

# 6 State space visualization

In this section we discuss the results of the graphical representation of the state spaces. Hereby a special graphical visualisation technique is used that was developed at the Technical University of Eindhoven to represent big state spaces. This visualisation technique seriously outperforms all contemporary

```
forall c:Customer,t:Nat.[true*.E0_c(t,c)]mu X.[!exists u:Nat.E6_c(u,c)]X
```

Figure 13: mCRL2 Specification: Every customer is served

techniques, which are generally restricted to at most a 100 states. This new visualisation technique allows to answer questions such as

- How many states exist in every part of the business process and business continuity process?
- Are areas in the state space independent in a way that there is no path between them?
- Is it possible to identify hot spots? Hot spots are groups of states that are passed through frequently in case of non-deterministic state sequences.

The visualization technique that has been introduced in [9] can visualize millions of states and transitions. It is further possible to inspect these states and transitions according to different criteria.

Currently, input state spaces are generated on the basis of behavioral specifications in mCRL2 from which state spaces in several formats can be produced. In these state spaces transitions are labeled with an action and states are labeled with a vector of data values. The basic idea underlying the visualization technique is to use a simplified representation in the form of a tree as a backbone for the entire structure. First the state space is layered using the distance of each node to the root. This distance is called the rank of the node. Then, the tree structure is obtained by clustering sets of states in each layer, such that for each set a unique path to the root is obtained. Each set of states is subsequently modeled using a disk shape in a three dimensional space. Finally, all disks are connected in a manner resembling cone trees, forming the shapes such as the ones in figures 14, 15, 16, 17, 18, and 19. Note that visual cues such as interactive motion, colors, lighting and transparency all add strongly to the three dimensional perception of the shapes. The still pictures in this print are by no means comparable to the onscreen images. After visualizing the state space as a tree shaped 3D-object we can use coloring to stress particular aspects of the state space. Typically, coloring can be induced by intrinsic properties such as the value of the transition label or state vector, or derived properties, such as the probability to visit a state during a random walk.

The visualization of the state space of the "Wertschöpfungskette" that handles only one customer is presented in figure 14. This graphics as well as all further graphics are created by the LTSview tool which is part of the mCRL2 suite available for download at the Technical University of Eindhoven. In figure 14, 161 states are shown. Some of them are highlighted by distinct colors. In addition to the states, we can find 182 transitions as well as 77 clusters that are ordered by the help of 30 ranks. In the given case pink (or dark) colored states mark a concrete state transition sequence.

The split into two main branches is caused by the fact that either one of two clerks is assigned to execute certain activities in the process. The green state indicates where a decision for one of the branches is taken. The four smaller branches show further alternative state sequences. From the point of view of the underlying business domain these branches can be explained by the credit worthiness check of a client by a clerk. This check can cause a client to be rejected and the check itself can be performed by different clerks. These alternatives lead to different state sequences in the visualization.

The visualization shows that there is no deadlock in the statespace. This is an important result regarding the BCP. A possible deadlock would have invalidated the solution. Another important result is that the visualization does not contain loop transitions. Loops would have invalidated the process likewise because it is assumed that the client considered is passing steadily through the process.

The following code shows the part where the first choice is taken. It is caused by the *assign_pclerk* statement.

The following sequence of process steps shows how the decision point is reached that divides into the two big branches as visualized in figure 15.

```
E0_C(0,0,C(0))
E0_Split1_C(0,0,C(0))
Split1_F1_C(0,0,C(0))
assign_pclerk_C(0,pc1,0,0)
Split1_F2_C(0,0,C(0))
```

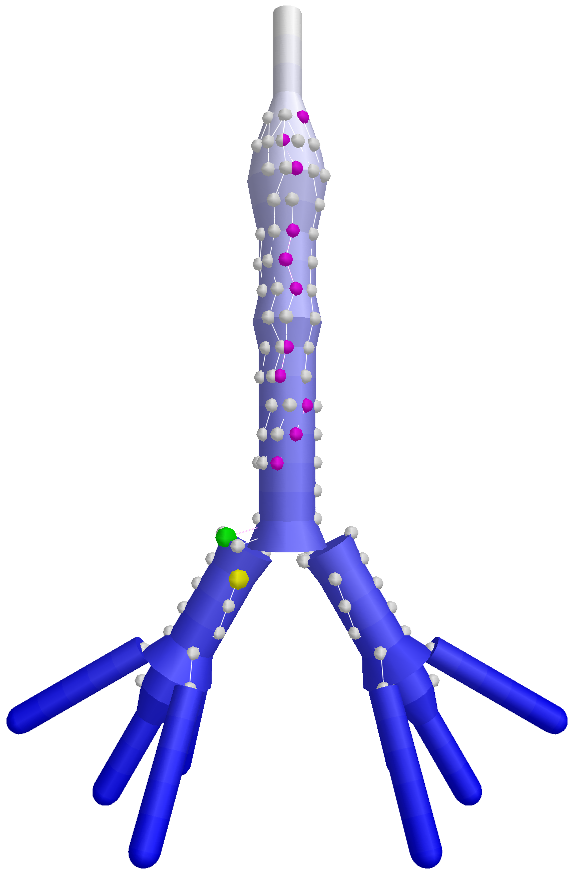
Figure 14: One customer in CVC - Without showing deadlocks

```
assign_pclerk_C(0,pc2,0,0))
assign_customer_C(0,0,C(0),0,0)
F1_getC_id(0,0,pC1,C(0))
release_Customer_C(3000,30,C(0))
release_pclerk_C(30,pc1)
F1_merge1_C(3000,30,C(0))
assign_customer_C(3000,30,C(0))
release_customer_C(4000,40,C(0))
release_pclerk_C(40,pc2)
F2_merge1_C(4000,40,C(0))
Merge1_E1a_C(4000,40,C(0))
data_grasp(4000,40,C(0))
E1a_F3a_C(4000,40,C(0))
assign_pclerk_C(40,pc1,40,30)
assign_pclerk_C(40,pc2,40,30)
```

In figure 15 deadlocks are identified by red (or dark) dots. These red dots can be found at the ends of the branches. Therefore, they indicate that the process terminates there as expected. Further deadlocks do not occur.

In figure 16 the same process is visualized, processing two clients now. As a consequence the number of states grows to 834 and the number of clusters to 717. 77 ranks are differentiated. This is because

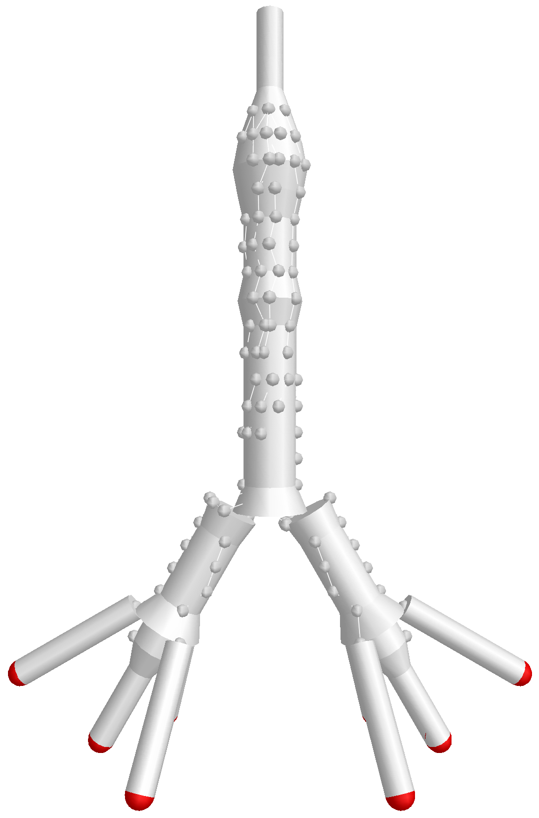

Figure 15: One customer in CVC - with read colored deadlock situation

there are now a number of alternatives realized that show how to handle two clients simultaneously in the process. The state sequences can be analyzed by the help of the tool LTSview in the same manner as already presented in figure 14. Therefore, we do not make it explicit here again. The only difference is a higher level of complexity in the state space.

The last three figures 17, 18 as well as 19 represent different variants of the Business Continuity Process. We like to mention that variant *B* is a modification of variant A.1. Variant A.2 did not succeed to fulfill the required capacity requirement. As a consequence the BCP was modified by relaxing the 4-eye-principle as a side constraint. So, it is possible that security requirements and business requirements are conflicting.

We can see that in figure 17 and 18 the visualizations are of a similar shape.

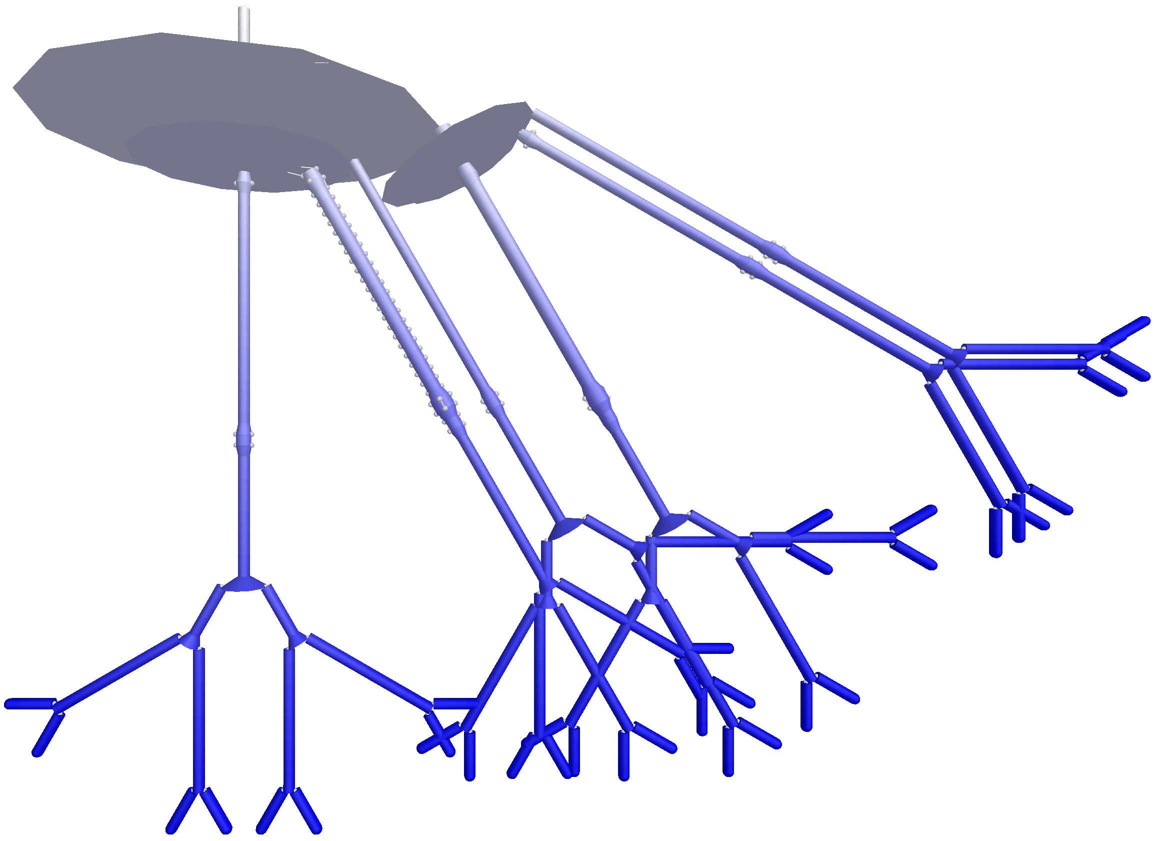

Figure 16: Two customers in CVC

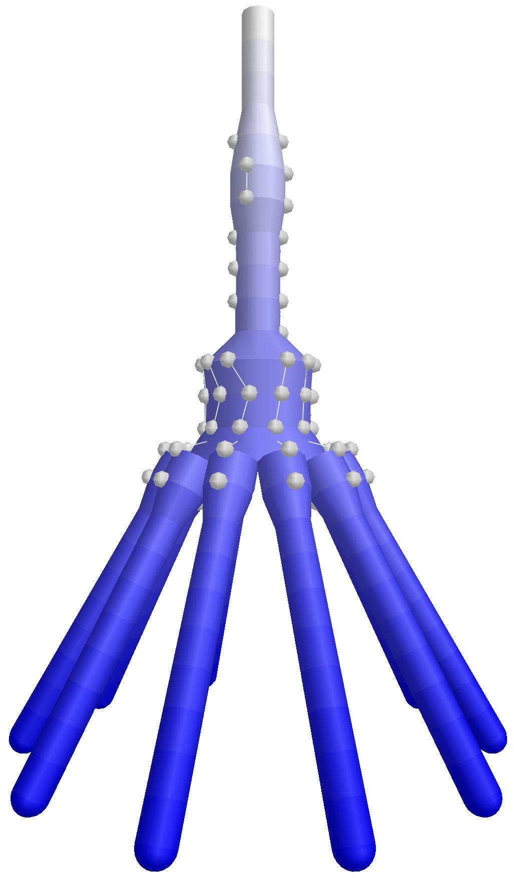

Figure 17: One customer in a Business Continuity Process (BCP) with Variant A.1

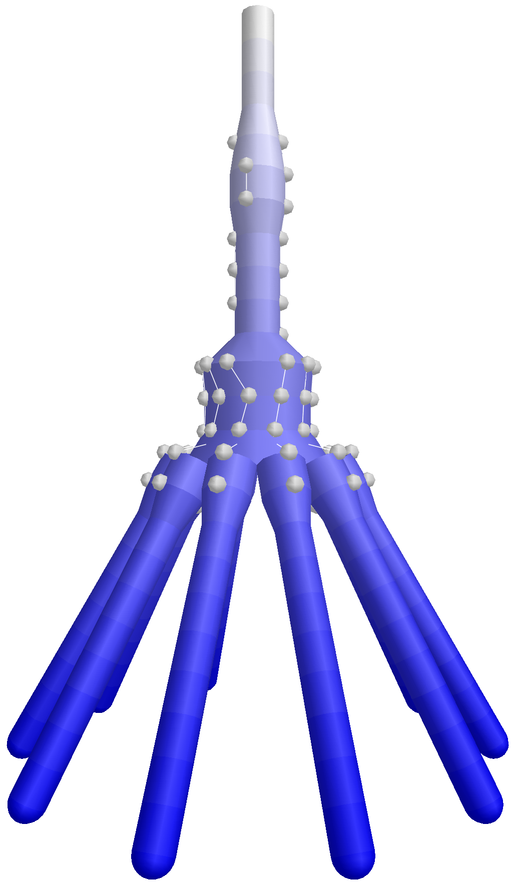

Figure 18: One customer in a Business Continuity Process (BCP) with Variant A.2

In figure 19 the variant B of the Business Continuity Process is presented. The structure is remarkable because it is ideal. There are no branches and no swellings. Therefore we do have an ideal BCP here that reduced the underlying business process at most. However, the disadvantage is that the 4-eye-principle is not longer implemented. It requires too much time and is therefore incompatible with the specific MTPD requirement.

Furthermore, there are no deadlock states visible in both figures except at the end of the branches. This indicates a sound termination of the processes. The processes do not get stuck in an unforeseen manner.

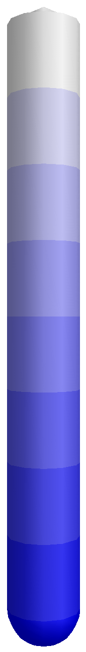

Figure 19: One customer in a Business Continuity Process (BCP) with Variant B

In this section we were able to present the possibilities of the tool LTSview that can visualize the state space of a business process. The shapes of the visualizations were able to point to decision points in the process, parallel activities and deadlock states. It is therefore very helpful to identify possible errors in the process specification and it is able to illustrate if a process is lean as it is the case in figure 19.

## 7 Related Work

There is a strong tendency to use formal techniques for process modelling. Below we mention studies that underline the need and mention the availability of several techniques.

Primarily, the work of Nemzow [17] discusses various strategies for protecting an organization from both natural and man-made disasters. IBM proposes [12] to use an integrated business continuity and resilience plan because traditional approaches to disaster planning have failed to keep organizations operational. Quirchmayr [22] presents an overview of business continuity management and addresses the question of how a system should be built in order to cope with a successful intrusion. Landry and Koger [16] show ten common disaster recovery myths. They claim that many organizations are unprepared or do make unrealistic assumptions. Tjoa, Jakoubi and Quirchmayr [26] discuss an approach of a business process modeling and simulation methodology that is risk-aware. To do so, they propose to advance the business impact analysis and the risk assement used today. A. and M. Zalewski, Sztandera and Ludzia [29] mention that the importance of business continuity and disaster recovery plans has grown considerably in the recent years. They explain that these plans are typically text documents and that exercising is still the main measure used to verify them.

A way to model these is by petri-nets, as suggested by van der Aalst [1]. Petrinets are not only suitable for modelling, but they can also be used for various verification purposes. Closer to our approach is the work of Puhlmann [18,21] who suggests to use the pi-calculus to explain the semantics of business processes. This is a little strange because classical data enlarged process algebras such as mCRL2 are much more mature for modelling and analysis, and the typical features of the pi-calculus do not seem to offer any additional value for business processes.

There are several other modelling formalisms. Thurner developed an own approach of how to formalize and verify event-driven process chains [25]. Koubarakis and Plexousakis [15] developed a formal framework for business process modeling and design. Their language permits to verify certain correctness properties. Boehmer developed a solution [3] to evaluate the performance of a business continuity management system according to the BS 25999.

Up till now, tool overviews for business processes do not mention the process algebraic tool suites because they are generally not used for business process simulation. See for instance the overview of Jansen-Vullers and Netjes [13].

# 8 Conclusions and Future Work

There are a couple of conclusions based on the result of this study: First, it is possible to define the semantics of an event driven process chain as it is used today for the purpose of business process modeling. Second, it is possible to use this semantics to check functional and non-functional properties of a business workflow in a fully automated fashion. Third, it is possible to simulate all possible state transitions to investigate the dynamic nature of a business process.

It is therefore possible to evaluate a CVC and a BCP as part of BCMS in an ex-ante way. By doing this, we can make a sound statement about the effectiveness and efficiency of a BCMS. We can further make sound statements about questions regarding the compliance of business processes with organizational policies. We claim that this will lead to a significant reduction of the BCMS development costs as well as a reduction of risks caused by non-functional BCPs and a reduction of insurance costs caused by a better understanding of the business continuity costs.

Future work will encompass the evaluation of further business cases and the ongoing development of the formal model.

# 9 Appendix

## 9.1 Process Specification

In the following the model for the critical business process and the three BCPs (Business Continuity Processes) is presented.

### 9.1.1 CVC

Here, the model of the “wertschoepfungskette” is presented. The time parameter was set at the beginning of each action. The new state space explorer selects the actions with the lowest value for the first parameter. By multiplying the time by 100 and adding the identity of the customer, it is assured that when all others things are equal, the first customer that arrived will be served first. This also applies to the notfallprocesskette.

```
% This mCRL file describes the "Wertschopfungskette".

sort Time=Nat;

% Split1-Merge1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

proc Split1(c:Customer)=
       sum t:Time.E0_Split1_r(100*t+id(c),t,c).
              sum n:Nat.Split1_F1_s(n,t,c).sum n:Nat.Split1_F2_s(n,t,c).Split1(c);

proc Merge1(c:Customer)=
       sum t:Time.F1_Merge1_r(100*t+id(c),t,c).sum u:Time.F2_Merge1_r(100*u+id(c),u,c).
                    sum n:Nat.Merge1_E1a_s(n,max(t,u),c).Merge1(c);

% The "Customer arrives" event E0 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

proc E0(c:Customer)=
       sum t:Time,n:Nat.E0_1(n,t,c).
       sum n:Nat.E0_Split1_s(n,t,c).
       E0(c);

% The "Identify customer" process F1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

map duration_F1:Time;
eqn duration_F1=30;

proc F1(c:Customer)=
       sum t:Time.Split1_F1_r(100*t+id(c),t,c).
       sum e:PClerk, u,u':Time.assign_pclerk_r(u',e,t,u).
       sum v,v':Time.assign_customer_r(100*v'+id(c),v',c,u',v).
       F1_getC_ID(100*v'+id(c),v',e,c).
       sum n:Nat.release_customer_s(n,v'+duration_F1,c).
       release_pclerk_s(v'+duration_F1,e).
       sum n:Nat.F1_Merge1_s(n,v'+duration_F1,c).
       F1(c);

% The "get Customer master data" process F2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

map duration_F2:Time;
eqn duration_F2=10;

proc F2(c:Customer)=
       sum t:Time.Split1_F2_r(100*t+id(c),t,c).
       sum e:PClerk, u,u':Time.assign_pclerk_r(u',e,t,u).
       sum v,v':Time.assign_customer_r(100*v'+id(c),v',c,u',v).
          % Perform task, but this is not really important, to measure througput.
       sum n:Nat.release_customer_s(n,v'+duration_F2,c).
       release_pclerk_s(v'+duration_F2,e).
       sum n:Nat.F2_Merge1_s(n,v'+duration_F2,c).
       F2(c);

% The "data grasped" event E1a %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

proc E1a(c:Customer)=
       sum t:Time.Merge1_E1a_r(100*t+id(c),t,c).
       data_grasped(100*t+id(c),t,c).
       sum n:Nat.E1a_F3a_s(n,t,c).
       E1a(c);

% The "Store information in a file" process F3a %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

map duration_F3a:Time;
eqn duration_F3a=5;

proc F3a(c:Customer)=
       sum t:Time.E1a_F3a_r(100*t+id(c),t,c).
       sum e:PClerk, u,u':Time.assign_pclerk_r(u',e,t,u).
          % Perform task, but this is not really important, to measure througput.
       release_pclerk_s(u'+duration_F3a,e).
       sum n:Nat.F3a_E1_s(n,u'+duration_F3a,c).
       F3a(c);

% The "information stored in file" event E1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

proc E1(c:Customer)=
       sum t:Time.F3a_E1_r(100*t+id(c),t,c).
       information_stored_in_file(100*t+id(c),t,c).
       sum n:Nat.E1_F3_s(n,t,c).
       E1(c);

% The "Check credit worthiness" process F3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

map duration_F3:Time;
eqn duration_F3=60;

proc F3(c:Customer)=
       sum t:Time.E1_F3_r(100*t+id(c),t,c).
       sum e:PClerk, u,u':Time.assign_pclerk_r(u',e,t,u).
       F3_check_credit_worthiness(100*u'+id(c),u',e,c).
       release_pclerk_s(u'+duration_F3,e).
       ((Customer_values.id(c)>70)->sum n:Nat.F3_E2a_s(n,u'+duration_F3,c)
% Condition to accept is value>70
                      <>sum n:Nat.F3_E2b_s(n,u'+duration_F3,c)).
       F3(c);

% The "Customer rejected " event E2a %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
proc E2a(c:Customer)=
       sum t:Time.F3_E2a_r(100*t+id(c),t,c).
       customer_accepted(100*t+id(c),t,c).
       sum n:Nat.E2a_F4_s(n,t,c).
       E2a(c);

% The "Customer rejected " event E2b %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

proc E2b(c:Customer)=
       sum t:Time.F3_E2b_r(100*t+id(c),t,c).
       customer_rejected(100*t+id(c),t,c).
       sum n:Nat.E6_s(n,t,c).
       E2b(c);

% The "Evaluate rating " process F4 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

map duration_F4:Time;
eqn duration_F4=15;

proc F4(c:Customer)=
       sum t:Time.E2a_F4_r(100*t+id(c),t,c).
       sum e:PPClerk, u,u':Time.assign_ppclerk_r(u',e,t,u).
          % Perform task, but this is not really important, to measure througput.
       release_ppclerk_s(u'+duration_F4,e).
       sum n:Nat.F4_E3_s(n,u'+duration_F4,c).
       F4(c);

% The "Bundled product chosen" event E3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

proc E3(c:Customer)=
       sum t:Time.F4_E3_r(100*t+id(c),t,c).
       bundled_product_chosen(100*t+id(c),t,c).
       sum n:Nat.E3_F5_s(n,t,c).
       E3(c);

% The "Create optimized contract" process F5 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

map duration_F5:Time;
eqn duration_F5=20;

proc F5(c:Customer)=
       sum t:Time.E3_F5_r(100*t+id(c),t,c).
       sum e:Supervisor, u,u':Time.assign_supervisor_r(u',e,t,u).
          % Perform task, but this is not really important, to measure througput.
       release_supervisor_s(u'+duration_F5,e).
       sum n:Nat.F5_E4_s(n,u'+duration_F5,c).
       F5(c);

% The "Contract created" event E4 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

proc E4(c:Customer)=
       sum t:Time.F5_E4_r(100*t+id(c),t,c).
       contract_created(100*t+id(c),t,c).
       sum n:Nat.E4_F6_s(n,t,c).
       E4(c);

% The "Print contract" process F6 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

map duration_F6:Time;
eqn duration_F6=5;

proc F6(c:Customer)=
       sum t:Time.E4_F6_r(100*t+id(c),t,c).
          % Perform task, but this is not really important, to measure througput.
       sum n:Nat.F6_E5_s(n,t+duration_F6,c).
       F6(c);

% The "Contract printed" event E5 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

proc E5(c:Customer)=
       sum t:Time.F6_E5_r(100*t+id(c),t,c).
       contract_printed(100*t+id(c),t,c).
       sum n:Nat.E5_F7_s(n,t,c).
       E5(c);

% The "Bank and customer sign form" process F7 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

map duration_F7:Time;
eqn duration_F7=30;

proc F7(c:Customer)=
       sum t:Time.E5_F7_r(100*t+id(c),t,c).
       sum e:Manager, u,u':Time.assign_manager_r(u',e,t,u).
          % Perform task, but this is not really important, to measure througput.
       release_manager_s(u'+duration_F7,e).
       sum n:Nat.F7_E6_s(n,u'+duration_F7,c).
       F7(c);

% The "Contract signed" event E6 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

proc E6(c:Customer)=
       sum t:Time.F7_E6_r(100*t+id(c),t,c).
       contract_signed(100*t+id(c),t,c).
       sum n:Nat.E6_s(n,t,c).
       E6(c);

% End tasks and events %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Customer %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

proc Customer(c:Customer)=
        sum t:Time,n:Nat.E0_3(n,t,c).Customer(c,t);

     Customer(c:Customer,t:Time)
      = sum t':Time,n:Nat.assign_customer_s(n,max(t,t'),c,t',t).
        sum u:Time.(u>=max(t,t')) -> release_customer_r(100*u+id(c),u,c).
        Customer(c,u)
      + sum t:Time.E6_r(100*t+id(c),t,c).delta;

sort Customer=struct c(id:Nat);
map  Customer_values:List(Nat);
eqn  Customer_values=[30,80,40,90,95,12,98,45,90,96,71,34];

proc Customers= Customer(c(0)) || Customer(c(1)) % || Customer(c(2)) || Customer(c(3))
              % || Customer(c(4)) || Customer(c(5))  || Customer(c(6)) || Customer(c(7))
              % || Customer(c(8)) || Customer(c(9)) % || Customer(c(10)) || Customer(c(11))
               ;

% Entering times of customers %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

map  Max_Delay:Time;
eqn  Max_Delay=1;

proc Entering_times_of_customers(id:Nat)=
      E0_2(100*10*id+id,10*id,c(id)).
      Entering_times_of_customers(id+1);

% Personel %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Realistic constellation, according to Christoph and Wolfgang.
%  2 Processing Clerk
%  2 post-processing Clerk
%  1 Supervisor
%  2 Manager

% Processing clerk %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

sort PClerk=struct pc1 | pc2;

proc PClerk(e:PClerk,t:Time)=
      sum t':Time.assign_pclerk_s(max(t,t'),e,t',t).
      sum u:Time.(u>=max(t,t')) -> release_pclerk_r(u,e).
      PClerk(e,u);

proc PClerks=PClerk(pc1,0) || PClerk(pc2,0);

% Post processing clerk %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

sort PPClerk=struct ppc1 | ppc2;

proc PPClerk(e:PPClerk,t:Time)=
      sum t':Time.assign_ppclerk_s(max(t,t'),e,t',t).
      sum u:Time.(u>=max(t,t')) -> release_ppclerk_r(u,e).
      PPClerk(e,u);

proc PPClerks=PPClerk(ppc1,0) || PPClerk(ppc2,0);

% Supervisor %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

sort Supervisor=struct sup1;

proc Supervisor(e:Supervisor,t:Time)=
      sum t':Time.assign_supervisor_s(max(t,t'),e,t',t).
      sum u:Time.(u>=max(t,t')) -> release_supervisor_r(u,e).
      Supervisor(e,u);

proc Supervisors=Supervisor(sup1,0);

% Manager %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

sort Manager=struct man1 | man2;

proc Manager(e:Manager,t:Time)=
      sum t':Time.assign_manager_s(max(t,t'),e,t',t).
      sum u:Time.(u>=max(t,t')) -> release_manager_r(u,e).
      Manager(e,u);

proc Managers=Manager(man1,0) || Manager(man2,0);

% Personel %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

proc Personel=PClerks || PPClerks || Supervisors || Managers;

% Tasks %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

proc Tasks(c:Customer)=E0(c)||Split1(c)||F1(c)||F2(c)||Merge1(c)||E1a(c)||F3a(c)||E1(c)||F3(c)||
                       E2a(c)||E2b(c)||F4(c)||E3(c)||F5(c)||E4(c)||F6(c)||E5(c)||F7(c)||E6(c);

% Actions %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

act  assign_pclerk_r,assign_pclerk_s,assign_pclerk_c: Time#PClerk#Time#Time;
     assign_ppclerk_r,assign_ppclerk_s,assign_ppclerk_c: Time#PPClerk#Time#Time;
     assign_supervisor_r,assign_supervisor_s,assign_supervisor_c: Time#Supervisor#Time#Time;
     assign_manager_r,assign_manager_s,assign_manager_c: Time#Manager#Time#Time;
     assign_customer_r,assign_customer_s,assign_customer_c: Nat#Time#Customer#Time#Time;
     release_pclerk_r , release_pclerk_s , release_pclerk_c : Time#PClerk;
     release_ppclerk_r , release_ppclerk_s , release_ppclerk_c : Time#PPClerk;
     release_supervisor_r , release_supervisor_s , release_supervisor_c : Time#Supervisor;
     release_manager_r , release_manager_s , release_manager_c : Time#Manager;
     release_customer_r , release_customer_s , release_customer_c : Nat#Time#Customer;
     E0_1 , E0_2 , E0_3 , E0_c : Nat#Time#Customer;
     E0_Split1_r , E0_Split1_s , E0_Split1_c : Nat#Time#Customer;
     Split1_F1_r , Split1_F1_s , Split1_F1_c : Nat#Time#Customer;
     Split1_F2_r , Split1_F2_s , Split1_F2_c : Nat#Time#Customer;
     F1_Merge1_r , F1_Merge1_s , F1_Merge1_c : Nat#Time#Customer;
     F2_Merge1_r , F2_Merge1_s , F2_Merge1_c : Nat#Time#Customer;
     Merge1_E1a_r , Merge1_E1a_s , Merge1_E1a_c : Nat#Time#Customer;
     E1a_F3a_r , E1a_F3a_s , E1a_F3a_c : Nat#Time#Customer;
     F3a_E1_r , F3a_E1_s , F3a_E1_c : Nat#Time#Customer;
     E1_F3_r , E1_F3_s , E1_F3_c : Nat#Time#Customer;
     F3_E2a_r , F3_E2a_s , F3_E2a_c : Nat#Time#Customer;
     E2a_F4_r , E2a_F4_s , E2a_F4_c : Nat#Time#Customer;
     F3_E2b_r , F3_E2b_s , F3_E2b_c : Nat#Time#Customer;
     F4_E3_r , F4_E3_s , F4_E3_c : Nat#Time#Customer;
     E3_F5_r , E3_F5_s , E3_F5_c : Nat#Time#Customer;
     F5_E4_r , F5_E4_s , F5_E4_c : Nat#Time#Customer;
     E4_F6_r , E4_F6_s , E4_F6_c : Nat#Time#Customer;
     F6_E5_r , F6_E5_s , F6_E5_c : Nat#Time#Customer;
     E5_F7_r , E5_F7_s , E5_F7_c : Nat#Time#Customer;
     F7_E6_r , F7_E6_s , F7_E6_c : Nat#Time#Customer;
     E6_s , E6_r , E6_c : Nat#Time#Customer;

     data_grasped : Nat#Time#Customer;
     information_stored_in_file : Nat#Time#Customer;
     rating_checked : Nat#Time#Customer;
     bundled_product_chosen : Nat#Time#Customer;
     contract_created : Nat#Time#Customer;
     contract_printed : Nat#Time#Customer;
     contract_signed : Nat#Time#Customer;
     customer_accepted : Nat#Time#Customer;
     customer_rejected : Nat#Time#Customer;

     F1_getC_ID , F3_check_credit_worthiness : Nat#Time#PClerk#Customer;
% System %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

init allow({ assign_pclerk_c ,
             assign_ppclerk_c ,
             assign_supervisor_c ,
             assign_manager_c ,
             assign_customer_c ,
             release_pclerk_c ,
             release_ppclerk_c ,
             release_supervisor_c ,
             release_manager_c ,
             release_customer_c ,
             E0_c ,
             E0_Split1_c ,
             Split1_F1_c ,
             Split1_F2_c ,
             F1_Merge1_c ,
             F2_Merge1_c ,
             Merge1_E1a_c ,
             E1a_F3a_c ,
             F3a_E1_c ,
             E1_F3_c ,
             F3_E2a_c ,
             E2a_F4_c ,
             F3_E2b_c ,
             F4_E3_c ,
             E3_F5_c ,
             F5_E4_c ,
             E4_F6_c ,
             F6_E5_c ,
             E5_F7_c ,
             F7_E6_c ,
             E6_c ,

             data_grasped ,
             information_stored_in_file ,
             rating_checked ,
             bundled_product_chosen ,
             contract_created ,
             contract_signed ,
             customer_accepted ,
             customer_rejected ,
             contract_printed ,

             F1_getC_ID ,
             F3_check_credit_worthiness
           },
      comm({
            assign_pclerk_r | assign_pclerk_s ->assign_pclerk_c ,
            assign_ppclerk_r | assign_ppclerk_s ->assign_ppclerk_c ,
            assign_supervisor_r | assign_supervisor_s ->assign_supervisor_c ,
            assign_manager_r | assign_manager_s ->assign_manager_c ,
            assign_customer_r | assign_customer_s ->assign_customer_c ,
            release_pclerk_r | release_pclerk_s ->release_pclerk_c ,
            release_ppclerk_r | release_ppclerk_s ->release_ppclerk_c ,
            release_supervisor_r | release_supervisor_s ->release_supervisor_c ,
            release_manager_r | release_manager_s ->release_manager_c ,
            release_customer_r | release_customer_s ->release_customer_c ,
            E0_1 | E0_2 | E0_3 ->E0_c ,
            E0_Split1_r | E0_Split1_s ->E0_Split1_c ,
            Split1_F1_r|Split1_F1_s->Split1_F1_c,
            Split1_F2_r|Split1_F2_s->Split1_F2_c,
            F1_Merge1_r|F1_Merge1_s->F1_Merge1_c,
            F2_Merge1_r|F2_Merge1_s->F2_Merge1_c,
            Merge1_E1a_r|Merge1_E1a_s->Merge1_E1a_c,
            E1a_F3a_r|E1a_F3a_s->E1a_F3a_c,
            F3a_E1_r|F3a_E1_s->F3a_E1_c,
            E1_F3_r|E1_F3_s->E1_F3_c,
            F3_E2a_r|F3_E2a_s->F3_E2a_c,
            E2a_F4_r|E2a_F4_s->E2a_F4_c,
            F3_E2b_r|F3_E2b_s->F3_E2b_c,
            F4_E3_r|F4_E3_s->F4_E3_c,
            E3_F5_r|E3_F5_s->E3_F5_c,
            F5_E4_r|F5_E4_s->F5_E4_c,
            E4_F6_r|E4_F6_s->E4_F6_c,
            F6_E5_r|F6_E5_s->F6_E5_c,
            E5_F7_r|E5_F7_s->E5_F7_c,
            F7_E6_r|F7_E6_s->F7_E6_c,
            E6_r|E6_s->E6_c
          },
          Entering_times_of_customers(0)||Customers||Personel||
          Tasks(c(0))
            ||Tasks(c(1))
            ||Tasks(c(2))
            ||Tasks(c(3))
            ||Tasks(c(4))
            ||Tasks(c(5))
            ||Tasks(c(6))
            ||Tasks(c(7))
            ||Tasks(c(8))
            ||Tasks(c(9))
          ));
```

### 9.1.2 BCP A.1, A.2 and B

The following code defines the behavior of A.1 and A.2 as well as B. To switch to the according behavior the flag on top of the model can be set to ”true” or ”false” to make the choice between A and B. Further on, once, A is choosen, proc F5a can be re-defined to switch between A.1 and A.2.

The following code snippet will make that clear for the choice of A.

```
map variantA:Bool;
eqn variantA=true;
```

The following code snippet will make that clear for the choice of B.

```
map variantA:Bool;
eqn variantA=false;
```

The following change in the code will determine if A.1 or A.2 is executed.

```
proc F5a(c : Customer)=F5a - alt1(c);
or
proc F5a(c : Customer)=F5a - alt2(c);
```

Below you will find the full model including all specifications for A.1, A.2 and B.

```
map variantA:Bool;
eqn variantA=false;  % Allows to choose between variant A or B. B has
                     % an additional pre-selection of customers.

% This mCRL file describes the "Notfallprozesskette"

sort Time=Nat;
```

```
% Check business value F0 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

map duration_F0:Time;
eqn duration_F0=60;

proc F0(c:Customer)=
       sum t:Time.E0_F0_r(100*t+id(c),t,c).
       sum e:PClerk, u,u':Time.assign_pclerk_r(u',e,t,u).
       F0_check_credit_worthiness(100*u'+id(c),u',e,c).
       release_pclerk_s(u'+duration_F0,e).
       ((Customer_values.id(c)>70)
                    ->sum n:Nat.F0_E0a_s(n,u'+duration_F0,c)   % Condition to accept is value>70
                    <>sum n:Nat.F0_E0b_s(n,u'+duration_F0,c)).
       F0(c);

% The "Accept customer" event E2a %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

proc E0a(c:Customer)=
       sum t:Time.F0_E0a_r(100*t+id(c),t,c).
       accept_customer(100*t+id(c),t,c).
       sum n:Nat.E0_Split_s(n,t,c).
       E0a(c);

% The "Reject customer" event E2b %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

proc E0b(c:Customer)=
       sum t:Time.F0_E0b_r(100*t+id(c),t,c).
       reject_customer(100*t+id(c),t,c).
       sum n:Nat.E6_s(n,t,c).
       E0b(c);

% Split-Merge %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

proc Split(c:Customer)=
       sum t:Time.E0_Split_r(100*t+id(c),t,c).
              sum n:Nat.Split_F1_s(n,t,c).sum n:Nat.Split_F2_s(n,t,c).Split(c);

proc Merge(c:Customer)=
       sum c:Customer,t:Time.F1_Merge_r(100*t+id(c),t,c).sum u:Time.F2_Merge_r(100*u+id(c),u,c).
                    sum n:Nat.Merge_E1a_s(n,max(t,u),c).Merge(c);

% The "Customer arrives" event E0 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

proc E0(c:Customer)=
       sum t:Time,n:Nat.E0_1(n,t,c).
       (variantA ->sum n:Nat.E0_Split_s(n,t,c)<>sum n:Nat.E0_F0_s(n,t,c)).
       E0(c);

% The "Identify customer" process F1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

map duration_F1:Time;
eqn duration_F1=30;

proc F1(c:Customer)=
       sum t:Time.Split_F1_r(100*t+id(c),t,c).
       sum e:PClerk, u,u':Time.assign_pclerk_r(u',e,t,u).
       sum v,v':Time.assign_customer_r(100*v'+id(c),v',c,u',v).
       F1_getC_ID(100*v'+id(c),v',e,c).
       sum n:Nat.release_customer_s(n,v'+duration_F1,c).
       release_pclerk_s(v'+duration_F1,e).
       sum n:Nat.F1_Merge_s(n,v'+duration_F1,c).
       F1(c);

% The "get Customer master data" process F2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

map duration_F2:Time;
eqn duration_F2=10;

proc F2(c:Customer)=
       sum t:Time.Split_F2_r(100*t+id(c),t,c).
       sum e:PClerk, u,u':Time.assign_pclerk_r(u',e,t,u).
       sum v,v':Time.assign_customer_r(100*v'+id(c),v',c,u',v).
          % Perform task, but this is not really important, to measure througput.
       sum n:Nat.release_customer_s(n,v'+duration_F2,c).
       release_pclerk_s(v'+duration_F2,e).
       sum n:Nat.F2_Merge_s(n,v'+duration_F2,c).
       F2(c);

% The "data grasped" event E1a %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

proc E1a(c:Customer)=
       sum t:Time.Merge_E1a_r(100*t+id(c),t,c).
       data_grasped(100*t+id(c),t,c).
       sum n:Nat.E1a_F4a_s(n,t,c).
       E1a(c);

% The "Accept customer" process F4a %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

map duration_F4a:Time;
eqn duration_F4a=60;

proc F4a(c:Customer)=
       sum t:Time.E1a_F4a_r(100*t+id(c),t,c).
       sum e:PClerk, u,u':Time.assign_pclerk_r(u',e,t,u).
       F4a_check_credit_worthiness(100*u'+id(c),u',e,c).
       release_pclerk_s(u'+duration_F4a,e).
       ((Customer_values.id(c)>70)
                    ->sum n:Nat.F4a_E2a_s(n,u'+duration_F4a,c)   % Condition to accept is value>70
                    <>sum n:Nat.F4a_E2b_s(n,u'+duration_F4a,c)).
       F4a(c);

% The "Accept customer" event E2a %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

proc E2a(c:Customer)=
       sum t:Time.F4a_E2a_r(100*t+id(c),t,c).
       accept_customer(100*t+id(c),t,c).
       sum n:Nat.E2a_F5a_s(n,t,c).
       E2a(c);

% The "Reject customer" event E2b %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

proc E2b(c:Customer)=
       sum t:Time.F4a_E2b_r(100*t+id(c),t,c).
       reject_customer(100*t+id(c),t,c).
       sum n:Nat.E6_s(n,t,c).
       E2b(c);

% The "Create standardized contract" process F5a %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

map duration_F5a:Time;
eqn duration_F5a=90;

proc F5a(c:Customer)=F5a_alt1(c);

proc F5a_alt1(c:Customer)=
       sum t:Time.E2a_F5a_r(100*t+id(c),t,c).
       sum e:PClerk, u,u':Time.assign_pclerk_r(u',e,t,u).
          % Perform task, but this is not really important, to measure througput.
       release_pclerk_s(u'+duration_F5a,e).
       sum n:Nat.F5a_E4a_s(n,u'+duration_F5a,c).
       F5a_alt1(c);

proc F5a_alt2(c:Customer)=
       sum t:Time.E2a_F5a_r(100*t+id(c),t,c).
       sum e:Manager, u,u':Time.assign_manager_r(u',e,t,u).
          % Perform task, but this is not really important, to measure througput.
       release_manager_s(u'+duration_F5a,e).
       sum n:Nat.F5a_E4a_s(n,u'+duration_F5a,c).
       F5a_alt2(c);

% The "Form filled" event E4a %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

proc E4a(c:Customer)=
       sum t:Time.F5a_E4a_r(100*t+id(c),t,c).
       form_filled(100*t+id(c),t,c).
       sum n:Nat.E4a_F7_s(n,t,c).
       E4a(c);

% The "Print contract" process F6 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

map duration_F7:Time;
eqn duration_F7=30;

proc F7(c:Customer)=
       sum t:Time.E4a_F7_r(100*t+id(c),t,c).
       sum e:Manager, u,u':Time.assign_manager_r(u',e,t,u).
          % Perform task, but this is not really important, to measure througput.
       release_manager_s(u'+duration_F7,e).
       sum n:Nat.F7_E6_s(n,t+duration_F7,c).
       F7(c);

% The "Contract signed" event E6 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

proc E6(c:Customer)=
       sum t:Time.F7_E6_r(100*t+id(c),t,c).
       contract_signed(100*t+id(c),t,c).
       sum n:Nat.E6_s(n,t,c).
       E6(c);

% End tasks and events %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Customer %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

proc Customer(c:Customer)=
        sum t:Time,n:Nat.E0_3(n,t,c).Customer(c,t);

     Customer(c:Customer,t:Time)
      = sum t':Time,n:Nat.assign_customer_s(n,max(t,t'),c,t',t).
        sum u:Time.(u>=max(t,t')) -> release_customer_r(100*u+id(c),u,c).
        Customer(c,u)
      + sum t:Time.E6_r(100*t+id(c),t,c).delta;

sort Customer=struct c(id:Nat) ;
map  Customer_values:List(Nat);
eqn  Customer_values=[30,80,40,90,95,12,98,45,90,96,71,34];


proc Customers= Customer(c(0)) || Customer(c(1)) || Customer(c(2)) || Customer(c(3))
             || Customer(c(4)) || Customer(c(5)) || Customer(c(6)) || Customer(c(7))
             || Customer(c(8)) || Customer(c(9))
             ;

% Entering times of customers %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
map   Max_Delay:Time;
eqn   Max_Delay=1;

proc  Entering_times_of_customers(id:Nat)=
        E0_2(1000*id+id,10*id,c(id)).
        Entering_times_of_customers(id+1);

% Personel %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Realistic constellation, according to Christoph and Wolfgang.
%  1 Processing Clerk
%  0 post-processing Clerk
%  0 Supervisor
%  1 Manager for 50% of his time.

% Processing clerk %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

sort  PClerk=struct pc1 | pc2;

proc  PClerk(e:PClerk,t:Time)=
       sum t':Time.assign_pclerk_s(max(t,t'),e,t',t).
       sum u:Time.(u>=max(t,t')) -> release_pclerk_r(u,e).
        PClerk(e,u);

proc  PClerks=PClerk(pc1,0) ;  % Only one clerk in the emergency case.

% Post processing clerk %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

sort  PPClerk=struct ppc1 | ppc2;

proc  PPClerk(e:PPClerk,t:Time)=
       sum t':Time.assign_ppclerk_s(max(t,t'),e,t',t).
       sum u:Time.(u>=max(t,t')) -> release_ppclerk_r(u,e).
        PPClerk(e,u);

proc  PPClerks=PPClerk(ppc1,0) || PPClerk(ppc2,0);

% Supervisor %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

sort  Supervisor=struct sup1;

proc  Supervisor(e:Supervisor,t:Time)=
       sum t':Time.assign_supervisor_s(max(t,t'),e,t',t).
       sum u:Time.(u>=max(t,t')) -> release_supervisor_r(u,e).
        Supervisor(e,u);

proc  Supervisors=Supervisor(sup1,0);

% Manager %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

sort  Manager=struct man1 | man2;

proc  Manager(e:Manager,t:Time)=
       sum t':Time.assign_manager_s(max(t,t'),e,t',t).
       sum u:Time.(u>=max(t,t')) -> release_manager_r(u,e).
        Manager(e,t+2*(max(u-t,0)));  % The manager has only 50% of his time available.

proc  Managers=Manager(man1,0); % Only one manager is available in the emergency case.

% Personel %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

proc  Personel=PClerks || PPClerks || Supervisors || Managers;

% Tasks %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

proc  Tasks(c:Customer)=E0(c)||F0(c)||E0a(c)||E0b(c)||Split(c)||F1(c)||F2(c)||Merge(c)||
                         E1a(c)||F4a(c)||E2a(c)||E2b(c)||F5a(c)||E4a(c)||F7(c)||E6(c);

% Actions %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

act  assign_pclerk_r,assign_pclerk_s,assign_pclerk_c: Time#PClerk#Time#Time;
     assign_ppclerk_r,assign_ppclerk_s,assign_ppclerk_c: Time#PPClerk#Time#Time;
     assign_supervisor_r,assign_supervisor_s,assign_supervisor_c: Time#Supervisor#Time#Time;
     assign_manager_r,assign_manager_s,assign_manager_c: Time#Manager#Time#Time;
     assign_customer_r,assign_customer_s,assign_customer_c: Nat#Time#Customer#Time#Time;
     release_pclerk_r,release_pclerk_s,release_pclerk_c:Time#PClerk;
     release_ppclerk_r,release_ppclerk_s,release_ppclerk_c:Time#PPClerk;
     release_supervisor_r,release_supervisor_s,release_supervisor_c:Time#Supervisor;
     release_manager_r,release_manager_s,release_manager_c:Time#Manager;
     release_customer_r,release_customer_s,release_customer_c:Nat#Time#Customer;
     E0_1,E0_2,E0_3,E0_c:Nat#Time#Customer;
     E0_Split_r,E0_Split_s,E0_Split_c:Nat#Time#Customer;
     E0_F0_r,E0_F0_s,E0_F0_c:Nat#Time#Customer;
     F0_E0a_r,F0_E0a_s,F0_E0a_c:Nat#Time#Customer;
     F0_E0b_r,F0_E0b_s,F0_E0b_c:Nat#Time#Customer;
     Split_F1_r,Split_F1_s,Split_F1_c:Nat#Time#Customer;
     Split_F2_r,Split_F2_s,Split_F2_c:Nat#Time#Customer;
     F1_Merge_r,F1_Merge_s,F1_Merge_c:Nat#Time#Customer;
     F2_Merge_r,F2_Merge_s,F2_Merge_c:Nat#Time#Customer;
     Merge_E1a_r,Merge_E1a_s,Merge_E1a_c:Nat#Time#Customer;
     E1a_F4a_r,E1a_F4a_s,E1a_F4a_c:Nat#Time#Customer;
     F4a_E2a_r,F4a_E2a_s,F4a_E2a_c:Nat#Time#Customer;
     F4a_E2b_r,F4a_E2b_s,F4a_E2b_c:Nat#Time#Customer;
     E2a_F5a_r,E2a_F5a_s,E2a_F5a_c:Nat#Time#Customer;
     F5a_E4a_r,F5a_E4a_s,F5a_E4a_c:Nat#Time#Customer;
     E4a_F7_r,E4a_F7_s,E4a_F7_c:Nat#Time#Customer;
     F7_E6_r,F7_E6_s,F7_E6_c:Nat#Time#Customer;
     E6_s,E6_r,E6_c:Nat#Time#Customer;
```

```
    data_grasped:Nat#Time#Customer;
    accept_customer:Nat#Time#Customer;
    reject_customer:Nat#Time#Customer;
    form_filled:Nat#Time#Customer;
    contract_signed:Nat#Time#Customer;

    F1_getC_ID,F4a_check_credit_worthiness,
    F0_check_credit_worthiness:Nat#Time#PClerk#Customer;

% System %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

init allow({ assign_pclerk_c,
             assign_ppclerk_c,
             assign_supervisor_c,
             assign_manager_c,
             assign_customer_c,
             release_pclerk_c,
             release_ppclerk_c,
             release_supervisor_c,
             release_manager_c,
             release_customer_c,
             E0_c,
             E0_Split_c,
             E0_F0_c,
             F0_E0a_c,
             F0_E0b_c,
             Split_F1_c,
             Split_F2_c,
             F1_Merge_c,
             F2_Merge_c,
             Merge_E1a_c,
             E1a_F4a_c,
             F4a_E2a_c,
             F4a_E2b_c,
             E2a_F5a_c,
             F5a_E4a_c,
             E4a_F7_c,
             F7_E6_c,
             E6_c,

             data_grasped,
             accept_customer,
             reject_customer,
             form_filled,
             contract_signed,

             F1_getC_ID,
             F4a_check_credit_worthiness,
             F0_check_credit_worthiness
           },
    comm({
            assign_pclerk_r|assign_pclerk_s->assign_pclerk_c,
            assign_ppclerk_r|assign_ppclerk_s->assign_ppclerk_c,
            assign_supervisor_r|assign_supervisor_s->assign_supervisor_c,
            assign_manager_r|assign_manager_s->assign_manager_c,
            assign_customer_r|assign_customer_s->assign_customer_c,
            release_pclerk_r|release_pclerk_s->release_pclerk_c,
            release_ppclerk_r|release_ppclerk_s->release_ppclerk_c,
            release_supervisor_r|release_supervisor_s->release_supervisor_c,
            release_manager_r|release_manager_s->release_manager_c,
            release_customer_r|release_customer_s->release_customer_c,
            E0_1|E0_2|E0_3->E0_c,
            E0_Split_r|E0_Split_s->E0_Split_c,
            E0_F0_r|E0_F0_s->E0_F0_c,
            F0_E0a_r|F0_E0a_s->F0_E0a_c,
            F0_E0b_r|F0_E0b_s->F0_E0b_c,
            Split_F1_r|Split_F1_s->Split_F1_c,
            Split_F2_r|Split_F2_s->Split_F2_c,
            F1_Merge_r|F1_Merge_s->F1_Merge_c,
            F2_Merge_r|F2_Merge_s->F2_Merge_c,
            Merge_E1a_r|Merge_E1a_s->Merge_E1a_c,
            E1a_F4a_r|E1a_F4a_s->E1a_F4a_c,
            F4a_E2a_r|F4a_E2a_s->F4a_E2a_c,
            F4a_E2b_r|F4a_E2b_s->F4a_E2b_c,
            E2a_F5a_r|E2a_F5a_s->E2a_F5a_c,
            F5a_E4a_r|F5a_E4a_s->F5a_E4a_c,
            E4a_F7_r|E4a_F7_s->E4a_F7_c,
            F7_E6_r|F7_E6_s->F7_E6_c,
            E6_r|E6_s->E6_c
          },
          Entering_times_of_customers(0)||Customers||Personel||Tasks(c(0))||Tasks(c(1))
                ||Tasks(c(2)) ||Tasks(c(3)) ||Tasks(c(4))||Tasks(c(5)) ||Tasks(c(6))
                ||Tasks(c(7)) ||Tasks(c(8)) ||Tasks(c(9))
                ));
```

## References

[1] W.M.P Aalst van der. Formalization and verification of event-driven process chains. *Information and Software Technology*, 41(10):639–650, 1999.

[2] Marco Alemanni, Grimaldi Alessia, Stefano Tornincasa, and Enrico Vezzetti. Key performance indicators for plm benefits evaluation: The alcatel alenia space case study. *Comput. Ind.*, 59(8):833–841, 2008.

[3] W. Boehmer. Survivability and business continuity management system according to bs 25999. In *SECURWARE '09*, volume 0, Athens/Glyfada, Greece, June 2009. IEEE Computer Society.

[4] Christoph Brandt, Thomas Engel, Wolfgang Boehmer, and Claude Roeltgen. Diskussionsvorschlag einer lösungsskizze zur behandlung von operationellen it-sicherheitsrisken nach basel ii auf der grundlage von anforderungen der credit suisse. In *Multikonferenz Wirtschaftsinformatik*, 2008.

[5] E. W. Dijkstra. Hierarchical ordering of sequential processes. In *Acta Informatica*, pages 1:115–138, 1971.

[6] T. Engels, J. F. Groote, M. van Weerdenburg, and T. Willemse. Search algorithms for automated validation. *J. Log. Algebr. Program*, (2009, to appear).

[7] J. F. Groote, A. Mathijssen, M. van Weerdenburg, and Y. S. Usenko. From $\mu$crl to mcrl2: Motivation and outline. In *Proc. Workshop Essays on Algebraic Process Cacluli (APC 25)*, pages 162:191–196, Bertinoro, Romagna, Italy, 2006. Electronic Notes in Theoretical Computer Science, Springer-Verlag.

[8] J. F. Groote and F. v. Ham. State space visualization. In *International Journal on Software Tools for Technology Transfer (STTT)*, volume 8, pages 77–91. Springer Berlin / Heidelberg, 2005.

[9] J.F. Groote, A.H.J. Mathijssen, M.A. Reniers, Y.S. Usenko, and M.J. van Weerdenburg. Analysis of distributed systems with mCRL2. In Michael Alexander and William Gardner, editors, *Process Algebra for Parallel and Distributed Processing*, pages 99–128. Chapman & Hall/CRC, 2008.

[10] J.F. Groote and M.A. Reniers. *Modelling and Analysis of Communicating Systems.* CRC Press, To appear. 2009.

[11] IBM. Panic slowly. integrated disaster response and built-in business continuity. ibm.com/itsolutions/uk/governance/businesscontinuity, 2006.

[12] M.H. Jansen-Vullers and M. Netjes. Business process simulation - a tool survey. In K. Jensen (ed.). 7th Workshop and Tutorial on the Practical Use of CPN 06, Volume 579 of DAIMI, pp. 77-96. Uni. of Arhus, Denmark, 2006.

[13] E. Kindler. Safety and liveness properties: A survey. *EATCS-Bulletin*, 53:268–272, June 1994.

[14] M. Koubarakis and D. Plexousakis. A formal framework for business process modelling and design. *Inf. Syst.*, 27(5):299–319, 2002.

[15] B. J. L. Landry and M. S. Koger. Dispelling 10 common disaster recovery myths: Lessons learned from hurricane katrina and other disasters. *J. Educ. Resour. Comput.*, 6(4):6, 2006.

[16] M. Nemzow. Business continuity planning. *Int. J. Netw. Manag.*, 7(3):127–136, 1997.

[17] F. Nestmann, U. & Puhlmann. *Business Process Specification and Analysis.* Chapman & Hall/CRC (ed) Alexander, Michael and Gardner, William, 1.st. edition, 2008.

[18] B. Ploeger and C. Tankink. Improving an interactive visualization of transition systems. In *Proceedings of the 4th ACM Symposium on Software Visualization (SoftVis 2008)*, pages 115–124. ACM, 2008.

[19] F. Puhlmann. Why do we actually need the pi-calculus for business process management? In W. Abramowicz & H. C. Mayr, editor, *9th Int. Conf. on Business Information Systems (BIS 2006)*, pages 77–89. Dept. of Information Systems, Poznań University of Economics, 2006.

[20] G. Quirchmayr. Survivability and business continuity management. In *ACSW Frontiers '04*, pages 3–6, Darlinghurst, Australia, 2004. Australian Computer Society, Inc.

[21] Raul Rodriguez Rodriguez, Juan José Alfaro Saiza, and Angel Ortiz Basa. Quantitative relationships between key performance indicators for supporting decision-making processes. *Computer in Industry*, 2008.

[22] A.-W. Scheer. *ARIS-Modellierungs-Methoden, Metamodelle, Anwendungen.* Springer, Berlin/Heidelberg, 2001.

[23] V. Thurner. *Formal fundierte Modellierung von Geschäftsprozessen – Geschäftsprozesse anschaulich und präzise dokumentieren.* ISBN 3-8325-0683-7. Logos Verlag, Berlin, 2004.

[24] S. Tjoa, S. Jakoubi, and G. Quirchmayr. Enhancing business impact analysis and risk assessment applying a risk-aware business process modeling and simulation methodology. In *ARES '08,*, pages 179–186, Washington, DC, USA, 2008. IEEE Computer Society.

[25] BSI (UK). Business continuity management system – part 1: Code of practice. ISBN 0580496015, 11 2006.

[26] BSI (UK). Business continuity management system – part 2: Specification. ISBN 9780580599132, 11 2007.

[27] A. Zalewski, P. Sztandera, M. Ludzia, and M. Zalewski. Modeling and analyzing disaster recovery plans as business processes. In *SAFECOMP '08*, pages 113–125, Berlin, Heidelberg, 2008. Springer-Verlag.