Dependability Model Transformation - A Stochastic Process Algebra Semantics for ZuverSicht Models

M. Riedl, J. Schuster, M. Siegle
University of the German Federal Armed Forces Munich
Department of Computer Science
{martin.riedl, johann.schuster, markus.siegle}@unibw.de

M. Blum, F. Schiller
Technische Universität München TUM
Department of Mechanical Engineering
Chair of Information Technology in Mechanical Engineering
{blum,schiller}@itm.tum.de

This paper presents a generic method for the automatic transformation from an application-oriented high-level dependability formalism to a low-level formal model. We start with ZuverSicht, a modelling framework for dependability and safety analysis of mechatronic systems. The target formalism is Stochastic Process Algebra (SPA), which has the advantage that the transformation can be done in a fully compositional manner. Transformation patterns for different dependability and safety aspects, such as failure and repair, error propagation, or failure with common cause are presented. The advantages of this approach are twofold: Firstly, the transformation defines a formal semantics for ZuverSicht models, and secondly, analysis of the low-level model can benefit from the efficient algorithms implemented in the SPA tool CASPA.

1 Introduction

Proper quantification of safety and reliability is a major issue in industrial automation. Plant operators are required by law to provide safety parameters (IEC61508 2000; ISO13849-1 2006), and the market requires the declaration of reliability parameters. Nowadays, machinery in industrial automation is made up of complex mechatronic devices and their associated complex safety functions. There are various methods for evaluating a system's safety, e.g. FMEA (Failure Mode and Effects Analysis), FTA (Fault Tree Analysis) (Vesely, Goldberg, Roberts, and Haasl 1981) or Markov models (Stewart 2009). A comparison of such methods can be found in (Rouvroye and van den Bliek 2002).

Markov models are among the most suitable methods for evaluating the safety of complex functions. However, working directly at the Markov chain level is not practicable for the human modeller, since model construction is very error-prone, especially if dynamic aspects such as error propagation, failures with common cause or different repair strategies need to be considered. Working with formalisms such as Stochastic Petri Nets (SPN) (Marsan, Balbo, Conte, Donatelli, and Franceschinis 1995) or Stochastic Process Algebra (SPA) (Hermanns, Herzog, and Katoen 2002) improves the situation but still requires an expert modeller. Therefore, there is a strong need for high-level application-oriented modelling languages with an automatic translation to the low-level Markov model which can then be analysed. Within the project ZuverSicht (Blum and Schiller. 2009) a user-friendly graphical notation was developed to specify all relevant properties of safety functions. ZuverSicht supports safety engineers providing the complete tool chain to perform specification work and calculate safety measures such as PFH (Probability of Failure per Hour). A database stores the specifications as safety patterns to enable re-use. The original ZuverSicht approach generates a Markov model of an overall safety function by means of the Kronecker sum of predefined component models. Afterwards, this system model is adapted by hard-coded operations to integrate all relevant information about diagnosis, error propagation, common cause failures and the specification of dangerous and safe system states. After the system model is parametrised with failure rates, test rates and repair rates, the user is able to calculate all relevant safety characteristics for the implementation of a predefined safety function.

This paper focuses on a new approach to generate calculation models from ZuverSicht models by means of a translation to SPA (instead of Kronecker sum and hard-coded manipulations). This has two advantages: Firstly, the translation to SPA provides a formal semantics for ZuverSicht models. Secondly, translating into SPA makes it possible to exploit the highly efficient analysis algorithms implemented in the tool CASPA (Kuntz, Siegle, and Werner 2004; Bachmann, Riedl, Schuster, and Siegle 2009), which is based on the symbolic data structure MTBDD and therefore able to cope with very large state spaces. This paper provides patterns and transformation rules that automatically transform a ZuverSicht model into a CASPA SPA model. However, the composition scheme presented here is generic, such that it could be abstracted and provided for other formalisms as well.

The paper is structured as follows: Sec. 2 presents an informal introduction to the ZuverSicht dependability domain, explaining how the components behave, and introducing the different dependability aspects of the formalism. Sec. 3 provides the necessary basics about the CASPA SPA. Sec. 4, the main part, presents the transformation rules and patterns. In Sec. 5, we discuss a case study of a simplified production cell. Our prototypical implementation is also briefly described. The paper finishes with a conclusion and some issues concerning future work.

2 The ZuverSicht Dependability Domain

In the ZuverSicht dependability domain, a system is modelled as a set of components which are subject to failure. In addition to the set of components, different dependability and safety aspects can be modelled. These will be described in the sequel:

2.1 Component behaviour

Fig. 1 depicts a component's behaviour. There are four possible states: The ok state (`OK`), the safe detected state (`SD`), the dangerous detectable state (`DD`) and the dangerous undetectable state (`DU`). The following events, inherent to the component, can occur: Event `safe detected` means that an error occurs, but this error is properly detected and does not lead to a dangerous situation. Event `dangerous detectable` is an error that leads to a dangerous situation, but this can afterwards be detected by a test routine `test`. Event `dangerous undetectable` is an error that is dangerous and cannot be detected. States `OK` and `SD` are considered safe, whereas states `DD` and `DU` are considered dangerous.

2.2 Safety Aspect: Safety Property and Hazards

The condition under which the overall system is considered safe is specified by a Reliability Block Diagram (RBD). As usual, an existing path through the

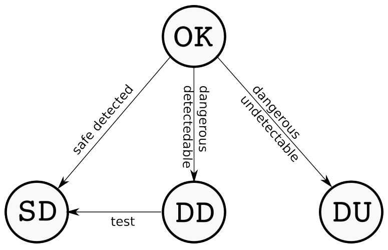


Figure 1. Internal behaviour of a component

RBD means that the system is safe, and a situation where no path through the RBD exists is dangerous. While the system is in such a dangerous situation, an external event can occur that brings the system as a whole into a hazardous state. Such an event represents the demand of the safety function. In ZuverSicht, these demand events occur randomly after an exponentially distributed time. After the occurrence of such a hazard, the system as a whole must be renewed in order to reach a clean state, where all components are again in the `OK` state.

In the example shown in Fig. 2, the overall system is safe if components C3 and C4 are both safe and at least one of the components C1 and C2 is safe.

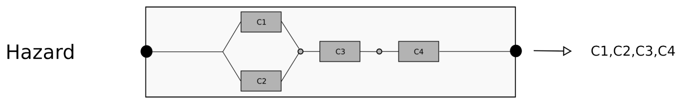


Figure 2. Example for a hazard condition

2.3 Dependability Aspect: Error Propagation

In ZuverSicht, error propagation may be caused by an arbitrary condition which is again specified by an RBD. The effect is that the affected components are no longer able to perform the self-test routine `test`.

In the example shown in Fig. 3, Component C3 would no longer be able to perform `test` if the condition specified by the RBD is satisfied.

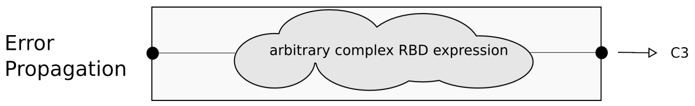


Figure 3. Error propagation affecting component C3

2.4 Dependability Aspect: Common Cause Failure

An external event, occurring at a certain rate, may be the common cause for the simultaneous failure of two or more components. As a result, the affected components transition to the `DD` or `DU` state. However, the common cause failure can only occur, if all affected components were originally in the `OK` state.

2.5 Dependability Aspect: Recovering by Repair

As soon as the first component moves to the `SD` state, the global repairman (GRM) is notified and will immediately start to repair the component. If other components move to their SD state while repair is in progress, they will also be repaired together with the previously failed components. When repair is finished, all components that are currently in the SD state move simultaneously to the OK state and the GRM becomes inactive again.

3 Syntax and Semantics of CASPA SPA

In this section, we give a short overview of the CASPA input language as far as it is needed for the transformation patterns given in Sec. 4. This language is a stochastic process algebra that has been extended recently to support both Markovian and timeless actions (Bachmann, Riedl, Schuster, and Siegle 2009). The basic building blocks of the language are sequential processes that can be combined by a parallel composition operator. To specify the behaviour of a sequential process, actions have to be used. An action in CASPA is a 2-tuple (l,v) consisting of a label l and a numerical value v. Two kinds of actions are distinguished:

- a Markovian action, denoted by (m,v), is driven by an exponentially distributed random variable with parameter v. In the sequel we also write $\xrightarrow{m,v}$.
- an immediate action, denoted by (*i,w*), defines a timeless probabilistic transition i with a probability given by its normalised weight w. Again, we may also write $\overset{i,w}{\dashrightarrow}$.

3.1 Definition of sequential processes

Fig. 4 shows an exemplary definition of a parametrised sequential process. Line (1) defines the name of the process (C1) and the name (state) and range (0,1) of its process parameter. One can identify a single process parameter with the state of the process, i.e. parameter state=i will be referenced as state i. The behaviour of the process

```
(1) C1(state [1]) :=
(2)  [state=0] -> (fail,3.0);C1(1) + (work,2.0);C1(0)
(3)  [state=1] -> (*C1_fail,1.0*);C1(1)
(4)  [*]       -> (*occ,1.0*);HAZC1(0)
```

Figure 4. Sequential process example 1

```
C1_ref(state [1]) := [state=0] -> (*C1_fail,1.0*);C1_ref(1)
```

Figure 5. Sequential process example 2

is defined in lines (2)-(4). Line (2) has a guard [state=0] meaning that the given transitions emanate from state 0. The choice operator (+) offers two concurring Markovian transitions: Either fail with rate 3 leading to state 1 or work with rate 2 returning to state 0 (self loop). In line (3) an immediate action C1_fail is given (a self-loop of state 1 with weight 1). Line (4) is valid for every state (guard [*]) and defines an immediate action occ with weight 1 that leads to state 0 of process HAZC1 (to be defined).

3.2 Parallel Composition

The crucial point for the transformation patterns given in Sec. 4 is that they have to be composed in parallel in a reasonable way. The CASPA language provides the following notation of parallel composition of the processes P1 and P2 with respect to a synchronisation set S resulting in a composed process P:

```
P:=P1|[S]|P2
```

That means that P consists of both P1 and P2 but with the constraint that the sequential processes are forced to perform all the actions in S together. The product state space consists of 2-tuples (x_1, x_2) where x_1 is a state of P1 and x_2 a state of P1. A formal approach can be found in (Hermanns, Herzog, and Katoen 2002; Bachmann, Riedl, Schuster, and Siegle 2009). We look at the parallel composition C of C1 (given in Fig. 4) and C1_ref (given in Fig. 5) starting in state $(0, 0)$:

```
C:=C1(0)|[C1_fail]|C1_ref(0)
```

Examples for possible transitions in the product state space are:

- $(0, 0) \xrightarrow{fail,3.0} (1, 0)$. Here C1 fails by its local transition fail, which does not affect C1_ref (i.e. C1_ref remains in its local state 0).
- $(1, 0) \overset{C1\_fail,1.0}{\dashrightarrow} (1, 1)$. Here C1 notifies C1_ref: C1 performs its self-loop while synchronously C1_ref changes its state.

3.3 Hiding actions

The CASPA input language also provides a hiding operator that can be used to hide internal behaviour. After the parallel composition of the example models C1(0)|[C1_fail]|C1_ref(0) the synchronisation set is not needed anymore, so the action C1_fail can be converted into an internal action named tau (so it cannot be used for further synchronisations). In the CASPA syntax one writes hide C1_fail in C1(0)|[C1_fail]|C1_ref(0). Note that for the transformations given in Sec. 4 it is not always allowed to hide the entire synchronisation set as some actions are needed for further synchronisations.

4 Transformation Patterns

In this section, we provide all transformation patterns and define a composition scheme needed to map a ZuverSicht model into a stochastic process algebra.

At first, in Sec. 4.1, the pattern for a ZuverSicht component is defined. In the following sections 4.2 - 4.6, all relevant patterns for the aspects are defined. In the last section 4.7, we provide rules how those patterns are composed together to obtain the overall process algebra model for the system.

4.1 Component Behaviour Pattern

The mapping of the ordinary ZuverSicht behaviour to process algebra is done directly. For the aspect of the system being in a safe or unsafe state the component processes has to provide information about its local safeness property. Assuming a component's process instance name C as shown in Fig. 6 (where □ is used as a placeholder for the rate parameters), the component pattern will be explained in the context of its aspects. At first the aspect of a global repairman is considered for a component, i.e. if a component reaches the safe detected state, a repairman must be notified (denoted by action nr). Whenever the repair process is completed, all component processes are notified using rd, which means that all processes that are currently in the safe detected state transition to their initial state and all the processes that were not in the safe detected state will remain in their current state. For each component, information can be retrieved concerning a certain property, e.g. if the component is currently safe or if it is functional. Therefore, each state has a number of self-loops corresponding to a certain property. The information whether or not a property is fulfilled can be obtained by synchronising with the corresponding self-loop actions. E.g. self-loops labelled C_safe, C_dang indicate if a component is currently safe or not. C_func and C_nonf indicate whether the system is functional, i.e. is in the OK-state, or not. If a common cause failure occurs then the affected components are notified by synchronising with C_dd,C_du, moving their state in either dangerous detectable or dangerous undetectable. The last important thing is that the system has an extra state that represents a system hazard. This means if a certain property is unsatisfied, the event occ converts all components independent of their current state simultaneously into the hazardous state. From the hazardous state, an event new moves all components simultaneously to their initial state. To reduce complexity and keep the reader

```
C(state [3]) :=
  [state=0] -> (sd,□);C(1)
  [state=0] -> (dd,□);C(2)
  [state=0] -> (du,□);C(3)
  [state=0] -> (*C_dd,1.0*);C(2)
  [state=0] -> (*C_du,1.0*);C(3)
  [state=2] -> (C_test,□);C(1)
  [state=0] -> (*C_safe,1.0*);C(0)
  [state=1] -> (*C_safe,1.0*);C(1)
  [state=2] -> (*C_dang,1.0*);C(2)
  [state=3] -> (*C_dang,1.0*);C(3)
  [state=0] -> (*C_func,1.0*);C(0)
  [state!=0] ->
      (*C_nonf,1.0*);C(state)
  [state=1] -> (*nr,1.0*);C(state)
  [state=1] -> (*rd,1.0*);C(0)
  [state!=1] -> (*rd,1.0*);C(state)
  [*] -> (*occ,1.0*);HAZC(0)

HAZC(state [0]) :=
  [state=0] -> (*new,1.0*);C(0)
```

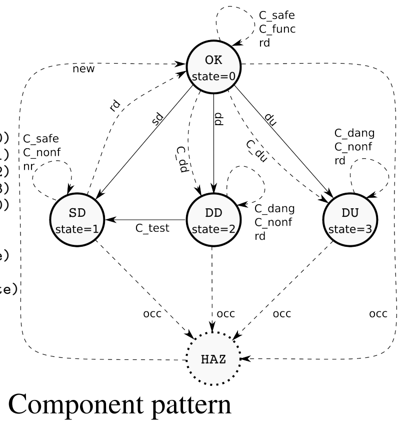


Figure 6. Component pattern

focused on the relevant parts, we henceforth do not show the sequential process part leading to the hazard state anymore, as this is the same as shown in the component pattern. This means that all subsequent process algebra definitions and figures are shortened by the hazard state and all involved actions.

4.2 Aspect Pattern of a Global Repairman

The first aspect introduced here is the global repairman aspect (see Fig. 7). If one component reaches the safe-detected state, the repairman (as the only repair process existing in that system) is notified nr. The repairman synchronises over the action by nr, i.e. starting to repair the system. After a certain time, the repair process is finished (represented by the Markovian action r) and the repairman signals that the repair process is complete with rd. All component process instances synchronise with rd simultaneously.

```
GRM(state [2]) :=
    [state=0] -> (*nr,1.0*);GRM(1)
    [state=1] -> (r,□);GRM(2)
    [state=2] -> (*rd,1.0*);GRM(0)
```

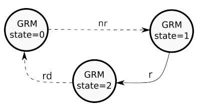


Figure 7. Global repairman pattern

4.3 Aspect Pattern of a Common Cause Failure

In ZuverSicht, a common cause failure affects two components. However, there are different error modes: either one component fails with dangerous undetectable and the other one with dangerous detectable, or both fail with the same dangerous failure type. The state S in Fig. 8 represents the situation where both components are in the OK state, while state U indicates that both components are nonfunctional. If one of these components becomes nonfunctional, the common cause process synchronises with an immediate action of the component (e.g. C1_nonf or C2_nonf) and can no longer perform the external common cause event. If one component becomes functional again (reaches the components OK state), the common cause process synchronises with C1_func or C2_func. If both components reach their OK state again, the common cause process transitions to its S state. The possible failure types are represented by Markovian actions labelled with the common cause identifier and the types of the dangerous failures: CC1_dddu, CC1_dudd, CC1_dddd, CC1_dudu. The rates for the Markovian common cause fail-

```
CC_CC1(state [3]) :=
  [state=0] ->
    (dudd,□);(*C1_du,1.0*);(*C2_dd,1.0*);CC_CC1(1)
  + (dddu,□);(*C1_dd,1.0*);(*C2_du,1.0*);CC_CC1(1)
  + (dudu,□);(*C1_du,1.0*);(*C2_du,1.0*);CC_CC1(1)
  + (dddd,□);(*C1_dd,1.0*);(*C2_dd,1.0*);CC_CC1(1)

  /* evaluation part */
  [state=0] ->
      (*C1_nonf,1.0*);CC_CC1(2)
    + (*C2_nonf,1.0*);CC_CC1(3)
  [state=2] ->
      (*C2_nonf,1.0*);CC_CC1(1)
    + (*C1_func,1.0*);CC_CC1(0)
  [state=3] ->
      (*C1_nonf,1.0*);CC_CC1(1)
    + (*C2_func,1.0*);CC_CC1(0)
  [state=1] ->
      (*C1_func,1.0*);CC_CC1(3)
    + (*C2_func,1.0*);CC_CC1(2)
```

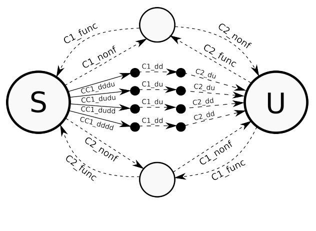


Figure 8. Common Cause Pattern

ures are calculated (according to (Blum and Schiller. 2009)) in the form

$$\lambda_{dudd} = \beta \cdot \min\{\lambda_{C1_{du}}, \lambda_{C2_{dd}}\},$$

```
C1_safe_ref(state [1]) :=
  [state=0] ->
      (*C1_dang,1.0*);(*pu,1.0*);C1_safe_ref(1)
  [state=1] ->
      (*C1_safe,1.0*);(*ps,1.0*);C1_safe_ref(0)
```

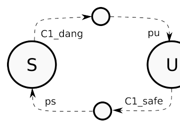


Figure 9. Pattern for retrieving the safeness aspect of a component

```
SBM(state [n]) :=
  [state<n] -> (*pu,1.0*);SBM(state+1)
  [state>0] -> (*ps,1.0*);SBM(state-1)
  [state=0] -> (*s,1.0*);SBM(state)
  [state>0] -> (*u,1.0*);SBM(state)
```

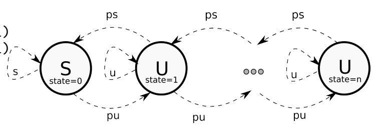


Figure 10. Pattern for composing evaluation processes in series

```
PBM(state [n]) :=
  [state<n] -> (*pu,1.0*);PBM(state+1)
  [state>0] -> (*ps,1.0*);PBM(state-1)
  [state<n] -> (*s,1.0*);PBM(state)
  [state=n] -> (*u,1.0*);PBM(state)
```

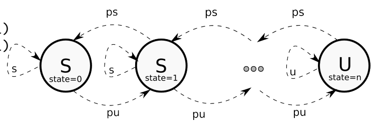


Figure 11. Pattern for composing evaluation processes in parallel

where β represents the ratio of common cause failures from the components failures.

4.4 Series-Parallel Evaluation Pattern

The ZuverSicht formalism allows arbitrary conditions to be defined in an RBD-style for the error propagation and hazard aspects. E.g. the safety property of a system can be defined by a series parallel RBD representing a logical formula. The RBD does not directly use a component instance, instead it uses a reference to a component instance. Therefore, arbitrary systems can be specified, i.e. also bridges or more complex networks (by using multiple references to the same component). In Fig. 9, a referencing process definition for a Component C1 is shown. It has to be synchronized with C1 using the synchronisation set C1_dang, C1_safe. The referencing process indicates whether the referenced process is in a safe state (ps) or in a dangerous state (pu). Other properties such as functional or nonfunctional, i.e. synchronising with C1_func, C1_nonf, C2_func, C2_nonf instead, can be expressed in a similar way. Assume now that n component references C1_ref and C2_ref are put in series. Then a counting process has to be introduced to determine (e.g. see Fig. 10) whether the series composition is safe or dangerous, e.g. if one component reference becomes unsafe (synchronized event pu has taken place) the series composition also exhibits an dangerous situation (self-loop u). The parallel composition is almost identical, only the s- and u-self-loops are different (see Fig. 11). To build nested structures a renaming process pattern is introduced (see Fig. 12), synchronizing with either a parallel or a serial process behaviour, that hides the actions s and u to the environment but synchronizing with them to provide the corresponding actions ps instead of s and pu instead of u to the outside.

```
RENAME(state [1]) :=
  [state=0] ->
      (*u,1.0*);(*pu,1.0*);RENAME(1)
  [state=1] ->
      (*s,1.0*);(*ps,1.0*);RENAME(0)
```

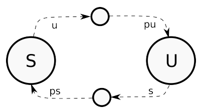


Figure 12. Pattern to allow nested parallel/series evaluation structures

4.5 Aspect Pattern for an Hazard

```
SYSHAZARD(state [2]) :=
  [state=0] ->
      (*pu,1.0*);SYSHAZARD(1)
  [state=1] ->
      (*ps,1.0*);SYSHAZARD(0)
  [state=1] ->
      (demand,□);(*occ,1.0*);SYSHAZARD(2)
  [state=2] ->
      (sysrep,□);(*new,1.0*);SYSHAZARD(0)
```

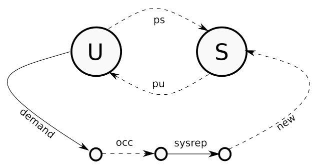


Figure 13. Syshazard aspect

As mentioned above, a ZuverSicht model has a certain global state that represents the system to be in a hazardous situation. The process instance of the nested evaluation patterns described before represents this safeness property of the system. To this evaluation process the hazard process (see Fig. 13) is composed. The Syshazard aspect which is synchronised within the evaluation process can either be in a unsatisfied state U or a satisfied state S. If the expression is unsatisfied, the Syshazard process can perform a Markovian action demand, afterwards synchronising over occ with all other processes, leading to the hazardous state. This hazardous state can only be left by performing a Markovian action sysrep and transitioning all processes into their initial state via action new.

4.6 Aspect of Error Propagation

An error propagation occurs if an arbitrary RBD expression is unsatisfied, affecting another component such that it is not able to perform its test any more. This evaluation takes place in the same form as introduced in Sec. 4.4. The evaluation process is composed with the error propagation process (Fig. 14), synchronising via the actions ps,pu. If the evaluation is satisfied, the self-loop to which the affected process has to be synchronised by Cx_test is enabled. If the evaluation is unsatisfied the error propagation process is in state U where no synchronised step over Cx_test is possible.

```
EPM_name(state [2]) :=
    [state=0] -> (*pu,1.0*);EPM_name(1)
    [state=1] -> (*ps,1.0*);EPM_name(0)
    [state=0] -> (Cx_test,1.0);EPM_name(state)
```

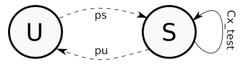


Figure 14. Error propagation aspect

4.7 Composition of the overall model

At first we provide a rule on how to compose an evaluation process used by the error propagation aspects and the hazard aspect. Evaluation processes contain reference processes to components providing the actions ps,pu evaluating if the given property is satisfied within a single component. In Fig. 15, one can see how parallel and series RBD expressions are composed in process algebra terms. An arbitrary evaluation expression can be substituted to the clouds. Their composition can be composed to a series, respectively a parallel behaviour model (denoted as SBM, resp. PBM), both synchronising over the actions ps,pu. Afterwards those actions can be made internal by applying the hide operator. Now, the composition exposes the actions s,u. Therefore the renaming process is composed to it performing the actions ps,pu whenever the actions s,u take place. Finally, the actions s,u can be hidden. Now, those series/parallel constructs provide the same actions to the outside and can be nested arbitrarily.

The model has a global hazard state into which each process has to move when a demand occurs by providing the immediate action occ. Reversely by the action new each process can move over into its initial state again. All processes in the system move over to the hazard state and back to their initial states simultaneously. Therefore one can find the actions occ and new in all following synchronisation sets.

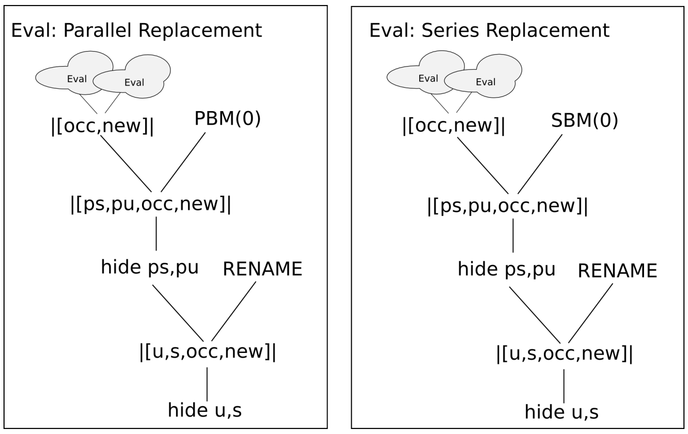


Figure 15. Evaluation composition scheme

The state of an evaluation is represented by its root process. A hazard aspect or an error propagation aspect can now be composed to it. In Fig. 16, one can also see the synchronisation set including ps,pu. Finally, these actions can be hidden. In Fig. 17, one

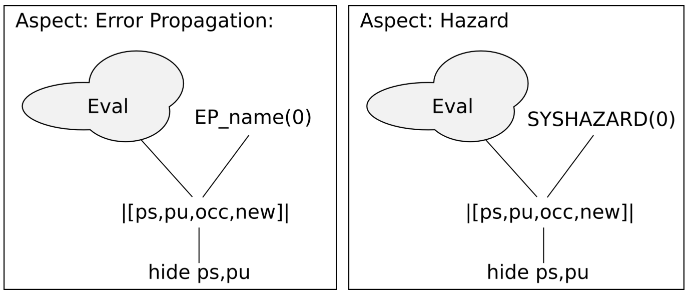


Figure 16. Aspect composition scheme

can see that for each component a process is instantiated and composed with other components processes. The repairman aspect repairs all component simultaneously, therefore the action rd must be part of the synchronisation set. Furthermore, all aspects are composed in parallel. In a final step, the components are composed with the aspects. Therefore, all synchronising actions used for the reference processes inside the evaluation structures are put into the synchronisation set: The actions Cx_du,Cx_dd of the affected component performed by a common cause failure, all Cx_test actions that are affected by an error propagation, rn,rd for the interaction with the repairman and finally of course occ,new to synchronise with the system hazard process.

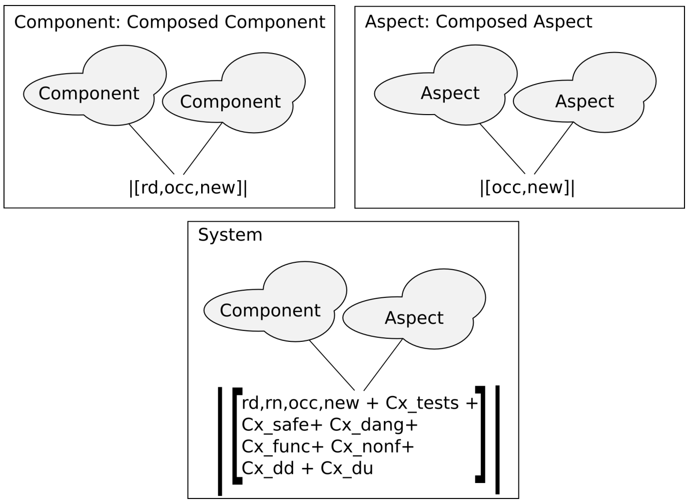


Figure 17. System composition scheme

5 Case Study

To demonstrate our approach, we use a typical safety relevant pneumatic application (see Fig. 18) consisting of two valves and a cylinder with an additional brake. The safety function is to stop the movement of the piston immediately and stay in position until another command occurs. The double-acting cylinder A1 can be moved and stopped by the 5-port/3-way valve S1, which is also used for the normal process. In order to stop the movement, both electromagnetic coils M1 and M2 are turned off and two return springs ensure that the valve returns to mid position, which blocks the entire air flow. To ensure a safety relevant stop, the 3-port/2-way valve S2 de-aerates the brake A2. The air tank Z1 and the non return valve V3 ensure the power supply of the pneumatic return spring of valve S2. The check valves with choke V1 and V2 enable a smooth movement and can be neglected in safety calculations. The cylinder is equipped with two sensors to detect the end positions of the piston (B1 and B2). These sensors enable a very good diagnosis of nearly all safety relevant components used in this application.

5.1 Textual Input Language

The graphical representation of the model and its aspects (see Fig. 19) can be serialised into textual form. For this purpose a domain specific textual language has been defined for the ZuverSicht language using the parser combinator library of Scala, a hybrid lan-

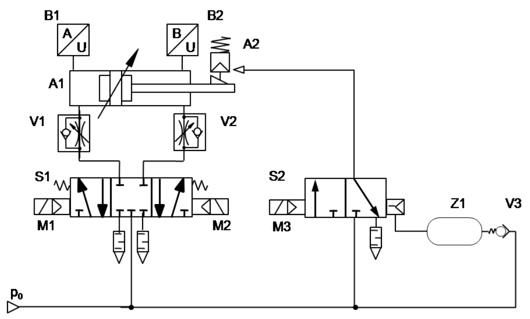


Figure 18. Typical safety relevant application

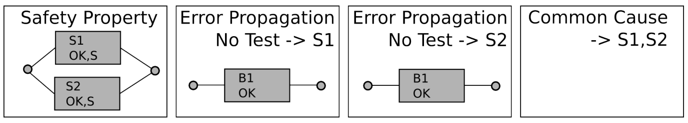


Figure 19. System aspects definition

guage that combines object orientation and a functional paradigm (see (Odersky, Spoon, and Venners 2008)). There, combinators can be defined in-place using an internal EBNF style domain language, without having to use parser generators. Each rule corresponds to a combinator as a basic building block (that is the parser for that rule). With the use of higher order functions, these building blocks can be combined until the whole language is covered. Without going into details, the components must be defined first. Then all aspects, starting with an RBD representing the evaluation formula of the property requested are specified. In Fig. 20, one can see our case study example transformed into its textual representation: At the beginning, the components `S1, S2, B1` are defined with rate parameters ($\lambda_{sd}, \lambda_{dd}, \lambda_{du}, \lambda_{test}$). Then the aspect of safety is given with rate parameters ($\lambda_{demand}, \lambda_{sysrep}$), implying a hazardous situation if its evaluation remains unsatisfied. A common cause failure affecting `S1, S2` with β is defined and lastly, two error propagation aspects are given such that, if `B1` is not functional, both `S1` and `S2` will not be able to perform a test.

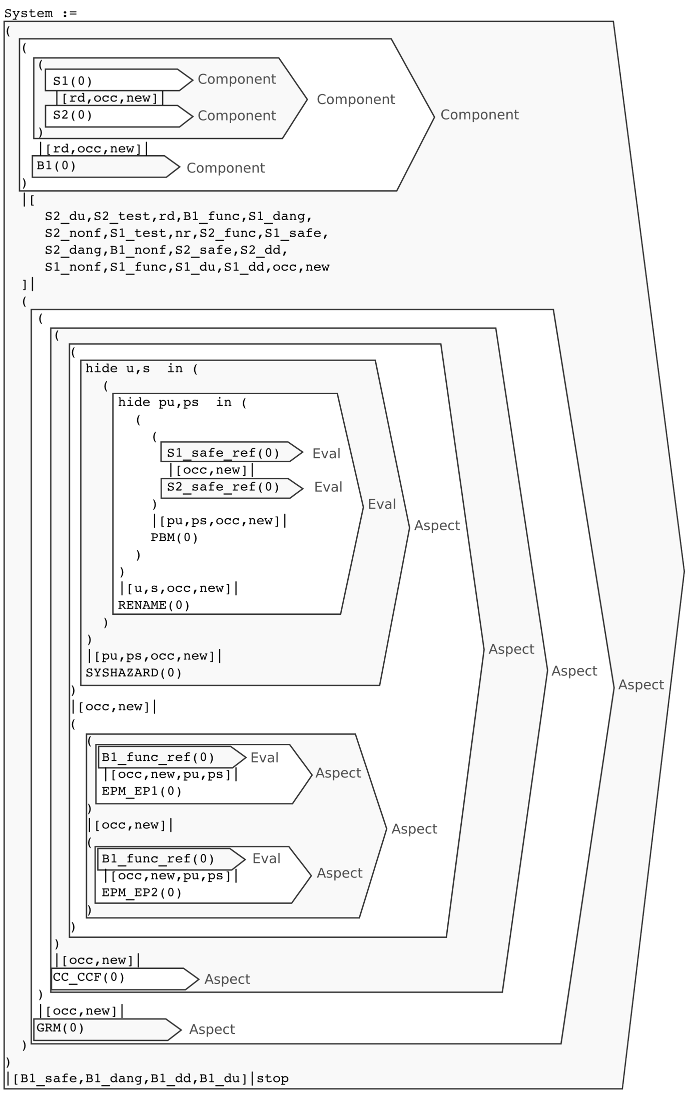
5.2 Internal Representation, Transformation and Analysis

Combinator operators compose those parsers and therefore define the language. The abstract model is built up whilst combinatory-parsing takes place. Next, the internal model is traversed and all transformation

```
Components:
  S1(7.0E-7, 7.0E-7, 7.0E-9,0.125, 1),
  S2(3.0E-7, 3.0E-7, 3.0E-9,0.125, 0.0416666666666667),
  B1(5.0E-6, 5.0E-6, 5.0E-7,0.125, 1)

       { 0 -> 1
         0 -s- S1 -> 1
         0 -s- S2 -> 1      } => HAZ(0.0416666666666667,0.125)
       {                    } => GRM(0.125)

CCF <- {                    } => CC(S1,S2,0.02)

EP1 <- { 0 -> 1
         0 -f- B1 -> 1      } => EP(S1)

EP2 <- { 0 -> 1
         0 -f- B1 -> 1      } => EP(S2)
```

Figure 20. Textual model

rules that have been described in Sec. 4 are applied, resulting in an abstract process algebra model that is transformed to the concrete CASPA syntax. In Fig. 21, one can see the resulting process algebra model. First, one can see that the process instantiations of the components `S1`, `S2` and `B1` are composed in parallel, synchronising over the actions `rd, occ, new`. Then, the subtree of the composed aspects is synchronised over the actions corresponding to the evaluation and aspect processes. The subtree includes the hazard process and its evaluation part, two error propagations `EP1`, `EP1` (both composed with one reference process representing the simplest possible evaluation formula including only one component), a common cause failure `CCF` and the global repairman `GRM`. Remaining unsynchronised immediate actions are inhibited by synchronisation with a stop process, in order to prevent the existence of timeless traps, which is done here for the actions `B1_safe,B1_dang,B1_dd,B1_du`. The



```
System :=
(
  (
   (
    S1(0)
     |[rd,occ,new]|
    S2(0)
   )
   |[rd,occ,new]|
   B1(0)
  )
  |[
     S2_du,S2_test,rd,B1_func,S1_dang,
     S2_nonf,S1_test,nr,S2_func,S1_safe,
     S2_dang,B1_nonf,S2_safe,S2_dd,
     S1_nonf,S1_func,S1_du,S1_dd,occ,new
  ]|
  (
   (
    (
     (
      hide u,s  in (
        (
         hide pu,ps  in (
           (
             (
              S1_safe_ref(0)
               |[occ,new]|
              S2_safe_ref(0)
             )
             |[pu,ps,occ,new]|
             PBM(0)
           )
         )
         |[u,s,occ,new]|
         RENAME(0)
        )
      )
      |[pu,ps,occ,new]|
      SYSHAZARD(0)
     )
     |[occ,new]|
     (
      (
       B1_func_ref(0)
        |[occ,new,pu,ps]|
       EPM_EP1(0)
      )
      |[occ,new]|
      (
       B1_func_ref(0)
        |[occ,new,pu,ps]|
       EPM_EP2(0)
      )
     )
    )
    |[occ,new]|
    CC_CCF(0)
   )
   |[occ,new]|
   GRM(0)
  )
)
|[B1_safe,B1_dang,B1_dd,B1_du]|stop
```

Figure 21. SPA model & composition hierarchy

textual process algebra model is then streamed into the CASPA tool (see (Hermanns, Herzog, and Katoen 2002; Bachmann, Riedl, Schuster, and Siegle 2009)).

5.3 Experimental Results

In this subsection, we present some numerical results for the model. The generated model has 1389 reachable states. After elimination of the immediate transitions used for the synchronisations, only 65 reachable states remain. The elimination is done in 8 symbolic elimination rounds (cf. (Bachmann, Riedl, Schuster, and Siegle 2009)). Our measure of interest is the average probability of failure per hour ($\mathtt{PFH}_{t_0}$) for a mission time of $t_0 = 20$ years. $\mathtt{PFH}_{t_0}$ is defined as

$$\mathtt{PFH}_{t_0} := \frac{r_{demand}}{t_0} \int_0^{t_0} \sum_{i \in \mathtt{Unsafe}} P_i(t)dt \qquad (1)$$

where $\sum_{i \in \mathtt{Unsafe}} P_i(t)$ is the probability that the system is in the state where the `demand` transition is enabled and r_{demand} is the global demand rate. In terms of Fig. 13, r_{demand} corresponds to the rate assigned to the Markovian action `demand` and the states where `demand` is enabled are those where `SYSHAZARD` is in state 1. Since CASPA does not directly provide means to calculate the integral in Eq. (1), we were restricted to transient analysis. We calculated $\sum_{i \in \mathtt{Unsafe}} P_i(t)$ by the measure `statemeasure demand SYSHAZARD (state=1)` from 0 to 20 years. The product

$$\mathtt{PFH}(t) := \sum_{i \in \mathtt{Unsafe}} P_i(t) \cdot r_{demand}$$

is given in Fig. 22 (left: first year, right: 0.5 to 20 years). Note that in the first year a quasi-jump occurs during the bootstrapping process. Using *Simpson's rule* for the supporting points $\{0, \frac{3}{365.25}, \frac{1}{2}, 1, 2, 3, \ldots, 20\}$ years we got the result $\mathtt{PFH}_{20} = 1.71063 \cdot 10^{-9}$ ¹/h. The ODE solver of the ZuverSicht framework calculated $1.71052 \cdot 10^{-9}$ ¹/h as the result, which fits quite well with our result (ZuverSicht under-approximates the result by calculating the flow out of the global hazard state). Looking at the steady-state, we obtain a very rough overestimation of $\mathtt{PFH}_{20}$: The CASPA result hereof is $1.7668 \cdot 10^{-7}$ ¹/h, which is also the result of the Zuversicht framework solver.

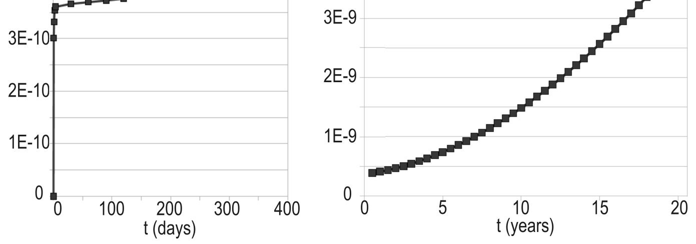


Figure 22. PFH(t)

6 Conclusion and Outlook

We have presented an approach to automatically derive stochastic process algebra models from ZuverSicht models in a compositional fashion by denoting a transformation semantics. Patterns in the form of sequential processes and rules on how to compose those processes, including the synchronisation needed, have been defined. Moreover, we demonstrated the automatism by a pneumatic safety application and analysed the relevant `PFH` measure. Still, immediate synchronisation leads to very large intermediate potential state spaces, so currently the elimination of immediate actions is the bottleneck of this approach. In the future, we plan to perform the elimination within the CASPA tool compositionally, therefore keeping the potential state space much smaller.

REFERENCES

Bachmann, J., M. Riedl, J. Schuster, and M. Siegle (2009). An Efficient Symbolic Elimination Algorithm for the Stochastic Process Algebra Tool CASPA. In *SOFSEM '09: Proceedings of the 35th Conference on Current Trends in Theory and Practice of Computer Science*, Berlin, Heidelberg, pp. 485–496. Springer LNCS 5404.

Blum, M. and F. Schiller. (2009). Effiziente Sicherheitsmodellierung in der Automatisierungstechnik. *SPS/IPC/DRIVES, Bender et al. (Hrsg.), VDE*, 189–197.

Hermanns, H., U. Herzog, and J.-P. Katoen (2002). Process algebra for performance evaluation. *Th. Comp. Sci. 274*(1-2), 43–87.

IEC61508 (1998-2000). *Functional Safety of electrical/electronic/programmable electronic safety-related Systems, Parts 1-7*. IEC.

ISO13849-1 (2006). *Safety of machinery - Safety-related parts of control systems - Part 1: General principles for design*. ISO.

Kuntz, M., M. Siegle, and E. Werner (2004). Symbolic Performance and Dependability Evaluation with the Tool CASPA. In M. Nunez, Z. Maamar, and F. Pelayo (Eds.), *Applying Formal Methods: Testing, Performance and M/E Commerce: FORTE 2004 Workshops, European Performance Engineering Workshop*, pp. 293–307. Springer, LNCS 3236.

Marsan, M. A., G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis (1995). *Modelling with generalized stochastic Petri nets*. Wiley.

Odersky, M., L. Spoon, and B. Venners (2008, November). *Programming in Scala: A Comprehensive Step-by-step Guide* (1st ed.). Artima Inc.

Rouvroye, J. and E. van den Bliek (2002). Comparing safety analysis techniques. *Reliability Engineering and System Safety 3*, 289–294.

Stewart, W. (2009). *Probability, Markov Chains, Queues, and Simulation*. Princeton University Press.

Vesely, W., F. Goldberg, N. Roberts, and D. Haasl (1981). *Fault Tree Handbook*. U.S. Nuclear Regulatory Commission.